

AMERICAN UNIVERSITY OF BEIRUT

IPSEC SCALABILITY: NETWORK DESIGN

by

NAREG LEVON ADALIAN

A thesis

submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
April 2016

AMERICAN UNIVERSITY OF BEIRUT

IPSEC SCALABILITY: NETWORK DESIGN

by
NAREG LEVON ADALIAN


Approved by:

Dr. Ayman Kayssi, Professor
Electrical and Computer Engineering



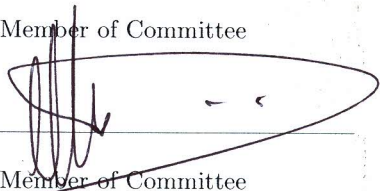
Advisor

Dr. Ali Chehab, Professor
Electrical and Computer Engineering



Member of Committee

Dr. Imad Elhadj, Associate Professor
Electrical and Computer Engineering



Member of Committee

Date of thesis defense: April 21, 2016

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: ___Adalian___ ___Nareg___ ___Levon___
Last First Middle

Master's Thesis Master's Project Doctoral Dissertation

I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

I authorize the American University of Beirut, **three years after the date of submitting my thesis, dissertation, or project**, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

Nareg April 26, 2016
Signature Date

Acknowledgements

I would like to thank our omnipotent, omniscient and omnipresent God for the strength, health and patience he has provided me to complete my thesis.

I am grateful for having Dr. Ayman Kayssi as my advisor, Dr. Imad Elhadj and Dr. Ali Chehab as co-advisors, without whom completing my thesis would have been impossible. Their moral support and instructive comments challenged, elevated and expanded my thinking horizon.

Last but not least, I would like to thank my family, relatives and friends whose constant support and noticeable presence throughout my road towards completing my work was always persistent. May God protect and bless them all throughout their lives.

An Abstract of the Thesis of

Nareg Levon Adalian for Master of Engineering
Major: Electrical and Computer Engineering

Title: IPsec Scalability: Network Design

IPsec is a point-to-point protocol that provides security between IP nodes and solves many network security problems by providing authentication, integrity and confidentiality. However, the point-to-point nature of IPsec does not allow the formation of a scalable IPsec network. Our aim is to design and implement an algorithm, k -Constrained Connected Dominating Set (k -CCDS), that constructs a scalable IPsec network, by creating a backbone which reduces the total number of Security Associations (SAs) needed to maintain a secure network, while satisfying the following three constraints: k -connected dominating set providing alternate disjoint paths, degree-constrained paths by limiting the number simultaneous SAs allowed on each node, and a shortest path by upper-bounding the cost of a path between two nodes (number of SAs a packet has to travel). The algorithm will form a backbone of IPsec gateways, where an SA exists between any two gateways that are directly connected and are part of the shortest path. When a node wants to communicate with another node (backbone or non-backbone), rather than forming an SA with the target node it will form an SA with the backbone IPsec gateway it is connected to which in turn will forward the packets securely through the backbone. Furthermore, k -CCDS can be used not only to form IPsec scalable networks, but also to construct any network architecture that requires the satisfaction of the provided three constraints. Our experimental results have shown that k -CCDS reduces the number of SAs required to construct scalable IPsec networks and the number of links needed to efficiently route packets in general scalable networks by 67% to 99.8% depending on the size of the network. Additionally, the proposed algorithm is proven to find a relaxed solution when a solution with the provided constraints does not exist.

Contents

Acknowledgements	v
Abstract	vi
1 Introduction	1
1.1 IPsec Overview	1
1.2 Motivation	3
2 Literature Review	5
2.1 VPN and IPsec Scalability	5
2.2 Degree Constrained Minimum Spanning Tree	8
2.3 Dominating Set Problem	9
2.3.1 Dominating Set Generalization	9
2.3.2 Vector Connectivity	9
2.3.3 Connected Dominating Set in Ad Hoc Wireless Networks	9
2.3.4 k -Connected k -Dominating Set in Ad Hoc Wireless Networks	10
2.4 Shortest Path Routing	12
3 System Model	14
3.1 Objectives	14
3.1.1 Connected Network	14
3.1.2 Shortest Path	15
3.1.3 Degree Constraint	15
3.1.4 Dominating Set	15
3.2 Definitions	15
3.3 System Design	17
3.3.1 Problem Statement	18
3.3.2 Network Architecture	19
3.3.3 IPsec Gateways	21
3.3.4 Example of a Scalable IPsec Network	22
3.4 Routing using k -CCDS	23
3.5 k -CCDS Security vs. Scalability	25

4	Algorithms	27
5	Proof of Algorithms	37
5.1	Problem Statement	37
5.2	Proof of Theorems	37
6	Experimental Results	42
6.1	Network Architecture	42
6.1.1	Scalable IPsec Networks	42
6.1.2	General Scalable Networks	42
6.2	Scalable IPsec Experimental Results	43
6.2.1	Simulation Parameters & Algorithm	43
6.2.2	Test Algorithm	45
6.2.3	Algorithm Efficiency	46
6.2.4	Experimental Results	46
6.3	General Scalable Networks	53
6.3.1	Simulation Parameters & Algorithm	53
6.3.2	Experimental Results	55
7	Conclusion and Future Work	62
A	Abbreviations	63

List of Figures

2.1	Example of Ad-Hoc Wireless Network [1]	10
2.2	Comparison of k -CDS Algorithms [2]	11
2.3	SPT Example with Degree Constraint Violation	12
2.4	SPT Example After Fixing Degree Constraint Violation	13
2.5	SPT Example with Minimum Penalty	13
3.1	Scalable IPsec Network Example	18
3.2	Count of Cost of a Path	20
3.3	Scalable Full-Mesh IPsec Network	23
4.1	Flow Chart of Algorithm 1	31
4.2	Flow Chart of Algorithm 2	34
6.1	Experiment 1 - Scalable IPsec SA Improvement	47
6.2	Experiment 1 - Scalable IPsec SA Improvement Scaled	47
6.3	Experiment 1 - Scalable IPsec % Reduction of SAs	48
6.4	Experiment 1 - Scalable IPsec Number of Backbones	49
6.5	Experiment 2 - Scalable IPsec SA Improvement	50
6.6	Experiment 2 - Scalable IPsec SA Improvement Scaled	50
6.7	Experiment 2 - Scalable IPsec % Reduction of SAs	51
6.8	Experiment 2 - Scalable IPsec Number of Backbones	52
6.9	Experiment 1 - General Networks Links' Improvement	55
6.10	Experiment 1 - General Networks Links' Improvement Scaled	56
6.11	Experiment 1 - General Networks % Reduction of Links	56
6.12	Experiment 1 - General Networks Number of Backbones	57
6.13	Experiment 2 - General Networks Links' Improvement	58
6.14	Experiment 2 - General Networks % Reduction of Links	59
6.15	Experiment 2 - General Networks Number of Backbones	60

List of Tables

6.1	Simulation Parameters for IPsec Networks	43
6.2	Total Number of SAs of Experiment 1	48
6.3	Total Number of SAs of Experiment 2	51
6.4	Traditional IPsec vs. Scalable IPsec	53
6.5	Simulation Parameters for General Networks	54
6.6	Total Number of Links of Experiment 1	57
6.7	Standard Deviation of Scalable Network Links of Experiment 1 . .	57
6.8	Standard Deviation of Traditional Network Links of Experiment 1	57
6.9	Standard Deviation of Number of Backbones of Experiment 1 . .	58
6.10	Total Number of Links of Experiment 2	59
6.11	Standard Deviation of Scalable Network Links of Experiment 2 . .	59
6.12	Standard Deviation of Traditional Network Links of Experiment 2	60
6.13	Standard Deviation of Number of Backbones of Experiment 2 . .	60
6.14	General Networks vs. Scalable General Networks	61

Chapter 1

Introduction

The network layer Internet Protocol (IP) provides an unreliable, best-effort, connectionless and insecure packet delivery service. It is an unreliable service because packets can get lost, be delivered out of order, get delayed and arrive as duplicates at the target host. It is a best-effort service because the network will do its best to deliver a packet. It is a connectionless service because routers do not have any state. It does not have any inherent security as it is easy to forge an IP packet's address, modify its content, replay old packets, and inspect the content of packets. In general, there is no guarantee that the received datagrams are from the claimed sender, contain the original intended data and that the original data was not inspected by an attacker (eavesdropper) during its transmission.

1.1 IPsec Overview

IP Security (IPsec) [3] protects IP packets by providing data origin authentication, data integrity, data confidentiality and protection from replay attacks [4]. IPsec protects IP datagrams and higher layer protocols (e.g. TCP and UDP) which can be exchanged between hosts, between network security gateways (e.g. firewalls or routers), or between hosts and security gateways. There exists a mandatory-to-implement suite of algorithms to ensure interoperability between different implementations. Shared keys are required by the security services of IPsec to provide authentication and/or confidentiality. The key management protocol is called Internet Key Exchange (IKE) [5].

IPsec supports two protocols which can be used separately or together:

- Authentication Header (AH): provides data origin authentication, data integrity and anti-replay protection. To identify an AH header, the IPv4 protocol field or the IPv6 next header field is set to 51.
- Encapsulating Security Payload (ESP): provides data origin authentication, data integrity, data confidentiality and anti-replay protection. To

identify an ESP header, the IPv4 protocol field or the IPv6 next header field is set to 50.

The IPsec protocols ESP and AH can be used in two modes:

- Transport mode: is used to protect the upper layer protocols such as TCP and UDP. The IPsec protocol, ESP or AH, header is inserted in front of the upper layer header and after the IP header. Transport mode is used to provide end-to-end security and hence it cannot provide identity protection because the IP address is sent in the clear.
- Tunnel mode: is used to protect the entire IP packet. In Tunnel mode, the IPsec protocol, ESP or AH, header is added in front of the original (inner) IP header and the new (outer) IP header is added in front of the IPsec header, where the new IP header can be the same or different from the original header based on the destination. The inner header specifies the communication endpoint whereas the outer IP header specifies the cryptographic or security gateway endpoint, which allows it to provide identity protection.

Every host in the network can have IPsec connections to multiple hosts using tunneling. To be able to distinguish between these connections and to create a communication contract between end-hosts, IPsec uses a communication identifier called Security Associations (SA) which are stored in the Security Association Database (SADB). An SA is used to identify the traffic that should be protected and with whom this protection is performed. Hence, both IPsec endpoints share the same SA. Every SA is identified by: Security Parameter Index (SPI) located in the IPsec header to uniquely identify an SA, IPsec protocol used and the destination address with which protection is performed. SA being unidirectional, two SAs are needed at each end-host to create an IPsec connection, one for inbound traffic and the other for outbound traffic.

To provide a secure communication channel between end-hosts, the users must be able to specify the required policies. The Security Policies (SP) are used to identify the level of security that should be applied to the traffic, in other words the authentication algorithms eg. HMAC-SHA-256, and encryption algorithms, eg. AES that should be used to protect the traffic. All the security policies are then stored in the Security Policy Database (SPD). Once a new traffic is obtained or generated at a host, the SPD is consulted to identify the traffic to be protected, how to protect it (through SPs), and with whom the protection is shared (through SAs). Based on the fields in the SPD, the traffic is either discarded, protected, or bypassed. SPD entries that define the action “protect” will point to an SA(s) that identify the state used to protect the packet.

The SPD of IPsec calls the Internet Key Exchange (IKE) [5] to create the SAs between two end-hosts. IKE operates inside the Internet Security Association &

Key Management Protocol (ISAKMP) [6] framework, which defines the packet formats and exchanged message construct, and is a combination of Oakley [7] and SKEME [8] protocols which define the steps two end-hosts must follow to create a shared and authenticated key. IKE SAs are initially created between the IKE peers which are the IPsec peers, and then the IPsec SA keys are derived from the IKE SA keys. IKE has its own policy settings through which it creates its SAs. It defines a policy in terms of protection suits, where each suit defines at least the encryption algorithm, the hash algorithm, the Diffie-Hellman group, and the method of authentication. These policies are stored in the IKE Policy Database and are weighted based on their order of preference.

The creation of the IKE SAs is known as Phase 1, which can operate in two modes: Main mode and Aggressive mode. After the creation of IKE SAs by Phase 1, the creation of IPsec SAs is known as Phase 2 which operates in Quick mode only.

1.2 Motivation

IPsec being a point-to-point protocol has limitations in constructing scalable secure networks which are the following:

- Two IPsec SAs are required between any two nodes to have secure communication between them in the network. Hence, we need a large number of SAs.
- Large memory is required to store all the security associations (SAs), which may cause a problem on limited memory-constrained devices.
- We have a large overhead in configuring and maintaining Phases 1 & 2 for each IPsec channel. Additionally, we have a frequent renewal of keys for each pair of IPsec peers.

Assuming that we have N nodes in a network, the total number of SAs needed to have a secure network is $2N(N-1)$. Consider a scenario of 1,000 nodes trying to establish a secure connection with each other using IPsec: every node needs to establish an end-to-end IPsec channel with every other node. Since each node needs two Security Associations (SAs), one for inbound traffic and another for outbound traffic, then every node needs to maintain 1,998 SAs to provide IPsec protection to these nodes. Furthermore, the total number of SAs needed to have a secure IPsec network is 1,998,000! After establishing a secure connection, the Internet Key Exchange (IKE) should periodically renew keys used to secure the communication to reduce the probability of someone compromising the secured system. An IPsec gateway or customer edges (CE) can be applied at the edge of our network, to provide security services to the clients connected to the IPsec

CE in a protected subnet. Because of the point-to-point nature of IPsec communication, IPsec gateways must be stable, highly reliable and scalable [9].

In this research, we propose the design of a backbone that allows the construction of a scalable IPsec network. Rather than establishing an SA as a communication channel between every two nodes, the corresponding non-backbone or backbone IPsec gateway will create a secure and reliable connection through the backbone IPsec gateways by creating a chain of SAs, which will connect the node securely to the requested backbone or non-backbone target IPsec gateway. We designed an algorithm, *k*-Constrained Connected Dominating Set (*k*-CCDS), that constructs a scalable secure network while satisfying the following constraints: minimize the size of the backbone, provide *k* alternate disjoint paths to overcome node or link failures, minimize the cost of a path through limiting the number of SAs traversed, and upper bound the number of SAs allowed on each node. Furthermore, *k*-CCDS is not limited to construct IPsec scalable networks only. It is a general algorithm that can be applied to construct any scalable network architecture that requires the satisfaction of the stated constraints.

The rest of the thesis is organized as follows. In Chapter 2 we review the previous work done in this domain. Chapter 3 presents our system model. Chapter 4 demonstrates and explains the algorithms that we have developed to construct our model. Chapter 5 includes the proofs that demonstrate the correctness of our algorithms. In Chapter 6 we run experiments using our model, verify its correctness and performance, and compare with existing models. Finally, Chapter 7 concludes the thesis work.

Chapter 2

Literature Review

The aim of this literature review is to give a general overview about some important concepts related to the thesis work.

2.1 VPN and IPsec Scalability

Francesco Palmieri in [10] analyzes and tests the scalability, weaknesses and strengths of two fast growing and highly performing VPN [11] architectures which provide secure connectivity through the insecure internet network. The first is IP layer overlay tunnel based VPNs, IPsec [3], and the second is peer network based VPNs, MPLS [12]. IPsec being a point-to-point protocol provides end-to-end strong security to its end-points through the insecure network, however it suffers from scalability issues, whereas the tag based MPLS architecture provides scalability meanwhile maintaining its high performance and isolated traffic through intelligent forwarding and tagging, but suffers from the customer's need of service provider's trust. IPsec has scalability issues because every host needs to have a tunnel with every other host in the network (number of hosts squared), which increases the computational overhead to maintain SAs and routing information, and this complexity and liability increases with the increase in the size of the network. Unlike IPsec, MPLS networks are scalable since customer edges (CE) do not need to know or maintain any information regarding other CEs or even run MPLS. This is possible because the provider edge (PE) is responsible in forwarding the packets through the provider core using smart routing techniques.

The author in [10] presents two extreme designs of IPsec networks; the first is the star topology where a single hub is responsible in relaying all the traffic between any two spokes by having each spoke have a pair of SAs toward the hub only, and the hub to have a pair of SAs with every spoke in the network. This solves the scalability problem but adds an overhead because every packet of every spoke has to be processed, encrypted and decrypted, by the hub which adds a lot of processing overhead and significant latency (due to maintaining all

the SAs and processing all network packets). The second scenario is the full mesh topology which has high security but a critical scalability problem, although it provides a lower latency compared with the star topology. MPLS VPN is highly scalable [10] because the provider core is transparent to the CEs and no tunneling, encryption, is used to keep connectivity between the communicating parties; it maintains scalability and security through creating private networks by using the inherent connectionless nature of TCP/IP networks. What the CEs need to know is simply the PE they should communicate with to connect with the network.

De Clercq et al. in [13] analyze in details the scalability issues that occur due to the architecture of MPLS VPN based networks. In layer 3 network-based VPNs, the intelligence and processing is concentrated in the PE router which must provide and maintain isolated contexts for different VPNs by first implementing Virtual Routers (VR), and second by creating tunnels between PE routers, where MPLS shows promising results. For large scalable networks consisting of thousands of VPNs, every PE has to maintain a large number of VRs, hence the need of a large number of routing and forwarding tables in addition to the presence of a large routing table needed to maintain internet connectivity. From the perspective of routing distribution and processing which is dependent on the number of VRs in PEs that belong to a certain VPN, whether the peer model [14] or the overlay model [12] is used, the absence of intelligent mechanisms that minimizes the propagation and processing of routing information will lead to an increase in the frequency of the update sessions and thus an increase in the negative impact on scalability. MPLS can solve the scalability problem that arises due to tunneling by multiplexing tunnels of VRs on two different PEs in a larger tunnel and thus the provider backbone needs to have the knowledge of these large tunnels only. Although MPLS VPN solves the scalability problem to a certain extent, it does not provide strong security because it does not use any cryptographic functions such as encryption and authentication.

Although MPLS VPN tries to implement security by creating private and isolated network, it doesn't provide any authentication and confidentiality [10], which becomes significantly important to protect the MPLS VPN from attackers when it is connected to the internet [13]. To incorporate control layer authentication, the authentication feature of BGP can be used to prevent the injection of wrong routes on the PEs by analyzing the label of the packets. In the absence of strong security, since the provider network cannot be fully trustable because it can depend on other SP networks to provide VPN to CEs, accidental misrouting can occur and thus different VPNs can access and eavesdrop each others data. Additionally, the lack of authentication prevents the filtering of data traffic coming from masqueraded or malicious PEs and CEs.

The addition of data layer confidentiality can take two forms [13]; customer based CE-CE end-to-end encryption and network based PE-PE encryption. CE-CE encryption alone does not provide protection from VPN resource spoofing because an intruder can inject malicious traffic on the CE-PE links and thus

the PE cannot know if the traffic is authentic or not, and will therefore forward the malicious packets. To prevent CE-PE eavesdropping and injections, further encryption and authentication can be added to this link which adds an extra processing overhead. Adding these cryptographic functions to maintain strong security degrades the scalability of the network because it adds processing due to the heavy cryptographic functions and an increase in memory consumption. PE-PE IPsec transport mode with IP tunneled MPLS packets is recommended by the authors in [15] which can reduce the number of SAs needed, but the need of CE-PE security to have a fully secure network is unavoidable, and thus additional SAs are negotiated. Furthermore, we can observe that if a certain PE-PE connection or PE is compromised, all the VPNs that have VRs in these PEs will become vulnerable because they use the same IPsec in transport mode to provide security.

Therefore, the authors in [13] showed that we have a tradeoff between scalability and strong security. As for the experiments performed in [10], the response time and performance of MPLS VPN, in terms of CPU and memory usage, was better compared to classical IPsec, however both showed similar results in terms of throughput. As for the scalability which is the most important factor, since the analysis has been done on 3 provider routers, 2 PEs and 4 CEs, it does not really capture the performance of MPLS VPN in scalable networks because scalable networks consist of hundreds and thousands of nodes.

Group VPN by Juniper [16] [17] and GET VPN (Group Encrypted Transport VPN) by Cisco [18] are extensions of IPsec protocol that allow the creation of secure scalable networks. Unlike traditional IPsec, nodes do not have a point to point tunnel or SA with every other network entity they want to securely communicate with. In Group VPN we have the concept of a trusted group, where all the members share a single group SA (SA TEK) and encrypt/decrypt the traffic of any other group member. Hence, the number of IPsec SAs needed is reduced from n^2 to 1, where n represents the number of nodes or members of the network. Thus Group VPN is a tunnel-less protocol because all the members share the same SA and no tunnel is created between any two group members. Furthermore, Group VPN provides tunnel header preservation. This is possible because of its tunnel-less nature where IPsec packets have the same inner and outer IP header, and thus the source and destination addresses are preserved. This allows Group VPN to route its encrypted packets using the already existing underlying network infrastructure rather than creating an overlay network. A Group VPN consists of a Group Controller and Group Members. During the first phase of creation of a group, every host that wants to join the group creates a unique IKE Phase 1 SA with the group controller as in traditional IPsec, and specifies the group it wants to register with after authenticating itself with the controller. The group controller in return sends all the necessary group keys, policies and SAs, using IKE SA, to the newly joining group member. This allows

the new group member to communicate with any other group member. As in traditional IPsec, we know that keys expire and to prevent the risk of leakage, the group controller periodically sends rekey messages to all group members, IKE Phase 2 shared by all members, which contains the new group SAs and keys before the expiration of the old group SA. As we can notice, we have two different types of traffic in Group VPN: control messages such as rekeys exchange between group controller and members which are encrypted and decrypted using Key Encryption Key (KEK), and data plane messages such as data packets between different group members which are encrypted and decrypted by group members using Traffic Encryption Key (TEK). In Group VPN [16] [17] when a group member cannot communicate with the group controller, it will stop forwarding traffic by default. This is known as fail-closed. In the fail-open scenario, only specific traffic is allowed to be forwarded through the group member without being encrypted until connectivity is restored, while the remaining obtained traffic is discarded. To add redundancy and prevent single point of failure [18], multiple group controllers or key servers can be used cooperatively to provide recovery from failure. The following are the most important services that are not supported by Group and GET VPN: high availability, Group VPN is recommended to be used in private networks, the whole group is compromised if a group member or the controller is compromised, overlapping Group VPNs can create mismatched SAs, and NAT traversal problems because of tunnel header preservation.

2.2 Degree Constrained Minimum Spanning Tree

The authors in [19] formulate the degree-constrained minimum spanning tree (DCMST) problem and provide three algorithms to generate a DCMST: primal method, dual method and branch & bound method (B&B). DCMST problem can be described as finding a MST such that the number of connections (in-degree and out-degree) of every node does not exceed a given degree constraint. The primal method is a modification of Prim's algorithm such that, the inclusion of any new edge into the tree does not violate the degree constraint. The dual method starts by creating a MST by running Prim's algorithm, and fixes the nodes that are violating the degree constraint by finding a replacement path that minimizes the penalty of the cost on the DCMST. The B&B method [20], calculates the lower bound for each node by finding the MST through exclusion and inclusion of edges, which is very similar to the traveling salesman problem. The B&B method converges to a general optimal solution, however it requires more storage compared to the primal and dual methods, and the running time increases rapidly as the number of nodes is increased.

2.3 Dominating Set Problem

2.3.1 Dominating Set Generalization

In [21], the authors formalize the dominating set generalization problem and provide an approximate algorithm in calculating the dominating set of a graph. The aim of the dominating set problem is to find a dominating set with minimal cardinality such that, every node in G is either in the dominating set or is connected to at least k_i vertices of the set. They propose an approximate algorithm that calculates the dominating set of a graph but it takes exponential time. To find a small dominating set the authors propose that, it is better to assume that vertices of high degree should belong to the dominating set with higher probability than vertices of small degree.

2.3.2 Vector Connectivity

The authors in [22] formalize the vector connectivity problem and provide variations of their algorithm to be applied on three different types of graphs: split graphs, co-graphs and trees. Their motivation for vector connectivity comes from challenges in domination, connectivity and information propagation in social and other networks. Vector connectivity is studied by the authors because it provides connectivity between parts of a graph via node-disjoint (alternate) paths. The vector connectivity problem is very similar to the dominating set problem with an additional constraint that a node that does not belong to the dominating set must have k node disjoint paths, without having any restriction on the length of the involved disjoint paths. The authors propose that, vector connectivity can be solved in polynomial time on split graphs, co-graphs and trees. It can be approximated within a factor of $\ln(n) + 2$ in polynomial time. This is possible because, vector connectivity can be recast as a particular case of minimum sub-modular cover problem which allows to apply classical approximation due to Wolsey. However, no general algorithm has been proposed that could work on any type of graph.

2.3.3 Connected Dominating Set in Ad Hoc Wireless Networks

Wu et al. [1] [23] propose an approximation algorithm that can quickly determine the dominating set (DS) in a given connected network because finding an exact solution is NP-hard. In addition to that, they discuss procedures to update and recalculate the dominating set when the underlying network topology changes and describe efficient routing using connected dominating set (CDS). A subset of vertices of a graph called gateways or backbone nodes, is a Dominating Set (DS) if every vertex not in the subset is adjacent to at least one vertex in the subset. The

main advantage of a connected dominating set based routing is that it centralizes the whole network into a small CDS sub-network, where only the backbone keeps the routing information and there is no need to recalculate the routing tables as long as there are no changes in the network topology. The proposed process called Marking Process decides which nodes will eventually belong to the DS, and extends it by introducing two rules through assigning unique id to nodes to solve the problem of having the whole graph as DS when graph is symmetric. Every vertex in the graph has a marker $m(v) \forall v \in V$ that can be either true or false. Initially all the nodes are unmarked (False). The nodes that we mark will eventually form the backbone of the DS of our graph.

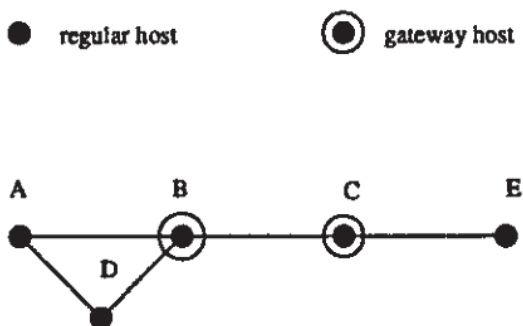


Figure 2.1: Example of Ad-Hoc Wireless Network [1]

In Figure 2.1 for example, B will be marked because its neighbors A and C or D and C are not connected. Similarly C will be marked because its neighbors B and E are not connected. The gateway host keeps information about gateway membership of the entire sub-network and local routing table. If shortest path routing is used, then the source sends a request to all its gateway neighbors to calculate the shortest path to the destination. They will reply by sending the minimum route length to the source. The source node will then pick the smallest one as its routing path.

2.3.4 k -Connected k -Dominating Set in Ad Hoc Wireless Networks

The authors in [2] propose and compare four construction protocols to obtain a CDS that maintains a certain degree of redundancy in the backbone for fault tolerance and routing flexibility; they allowed every node to have different paths to the backbone to avoid loss of important data during link or node failures. A node set is k -dominating if every vertex is either in the set or has k connections to the DS. Hence, the aim is to obtain a k -CDS as a virtual backbone that can survive failures of at least $k-1$ nodes. Four construction algorithms are proposed, which

are localized and rely only on neighborhood information because they are low cost and quickly converging. The time-complexity to verify the k -connectivity, k -node disjoint paths, of a graph is $O(k^2|E||V|)$.

Localized CDS algorithms are either probabilistic or deterministic. The first two algorithms, k -Gossip and k -Grid, have a probabilistic approach which incurs very low overhead and maintains a CDS with high probability, but produces a relatively large backbone and highly depends on network parameters. The third algorithm, k -CDS, is a deterministic algorithm which guarantees a CDS in connected networks because it uses smart selection methods, however it has a high computation/construction cost: Node v has a non-backbone status if for any two neighbors u and w , k node disjoint replacement paths exist that connect u and w via several intermediate nodes with high priorities than v . The fourth construction protocol is a hybrid paradigm that enables 1-CDS algorithm to construct k -CDS with high probability in dense networks and does not depend on any network parameters. It makes the migration process simpler and easier when extending an existing CDS algorithm to k -CDS compared to other construction algorithms which need to modify their CDS construction algorithm and hence make it more complex to obtain a k -CDS. It randomly partitions the network into k sub-networks with different colors (probabilistic approach) and applies a traditional CDS algorithm to each sub-network (deterministic part). Then joins the k -colored backbones forming a k -CDS: Node v has a non-backbone status if for any two neighbors, u and w , a replacement paths exist that connect u and w via several intermediate nodes (if any) with the same color as v and with higher priority than v . The following table provided by the authors, summarizes the comparison between the four k -CDS construction algorithms:

<i>Algorithm</i>	<i>Guarantees k-CDS</i>	<i>Expected Backbone Size</i>	<i>Communication Rounds</i>	<i>Message Size</i>	<i>Computation Cost</i>
<i>k-Gossip</i>	No	np_k	0	N/A	$O(1)$
<i>k-Grid</i>	No	$O(AB_k)$	1	$O(1)$	$O(\Delta)$
<i>k-Coverage</i>	Yes	$O(k).OPT$	2	$O(\Delta)$	$O(k.\Delta^4)$
<i>CBCC</i>	No	$O(1).OPT$	2	$O(\Delta)$	$O(k.\Delta^3)$

Figure 2.2: Comparison of k -CDS Algorithms [2]

2.4 Shortest Path Routing

The authors in [24] [25] propose an algorithm to construct a Bluetooth scatternet spanning given sensor nodes such that the total energy consumption during a single round of data transfer from every sensor node to the base station is kept to a minimum by making the total size of the scatternet as small as possible. The main difference between the design in [24] and [25] is that, in [25] the devices are capable of power control hence the energy needed for transmission depends on the distance (cost of an edge is not necessarily a unit cost) whereas in [24] the devices are not capable of power control hence it is independent of the distance between nodes (the edges have unit cost). The algorithm in [25] starts by forming a shortest path tree from the source using Dijkstra’s single source shortest path algorithm [26]. However, after obtaining the SPT the degree constraint of some nodes will not be satisfied. The algorithm then searches for the nodes that are violating the degree constraint. For every violating node, the first algorithm checks for every child if they can be connected to some other node, with minimal penalty, whose degree constraint is not violated when this new connection is added and if the connection is not part of the original SPT. Then for every child the connection that has the minimum penalty is selected and from all the children of the violating node the one with the minimal penalty among the minimal penalties is chosen to be disconnected from the violating node and connected to the newly selected node. The second algorithm tries to minimize this penalty by checking the descendants of the chosen child if they can be reconnected with a shorter path (less cost) to the SPT without violating a degree constraint. This whole process is repeated until the degree constraint of every node is satisfied. As

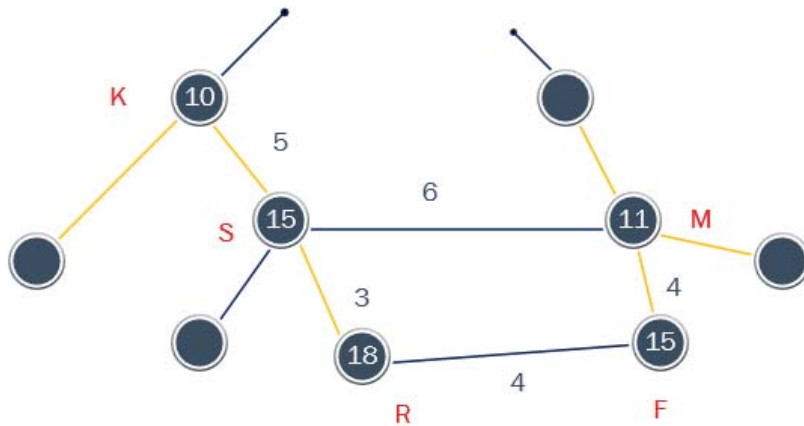


Figure 2.3: SPT Example with Degree Constraint Violation

an example assume that after running Dijkstra’s algorithm we obtain the SPT shown in Figure 2.3 where the yellow edges are part of the SPT. Also assume that

node K is violating the degree constraint. Next assume that connecting S to M

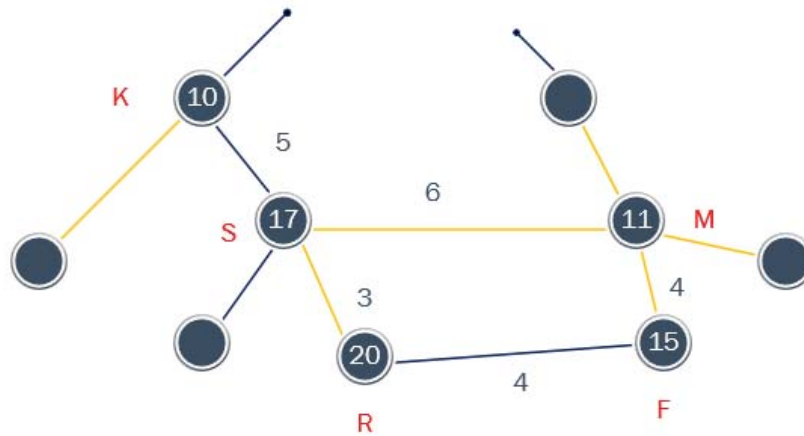


Figure 2.4: SPT Example After Fixing Degree Constraint Violation

minimizes the penalty we obtained in Figure 2.4, where the label of S increased by 2 and so did the label of R. Since the penalty of node R can be minimized by

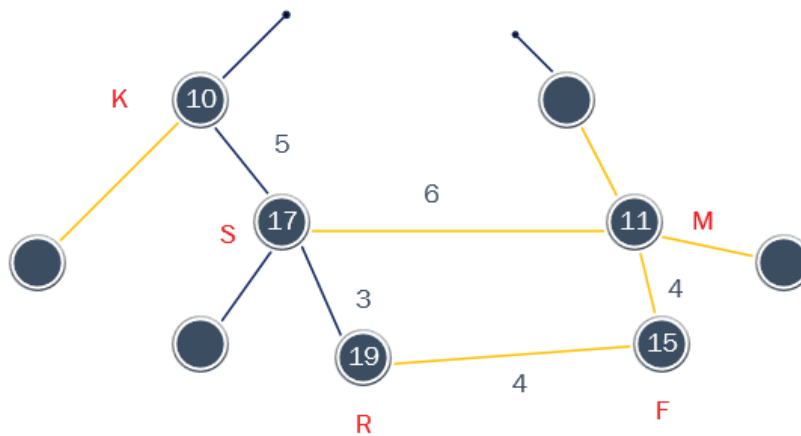


Figure 2.5: SPT Example with Minimum Penalty

connecting it to F, we finally obtain an increase in the label of S by 2 and the label of R by 1 rather than 2 as shown in Figure 2.5.

Chapter 3

System Model

In IPsec networks, any two hosts that want to communicate securely with one other have to create an end-to-end IPsec connection by creating a pair of IPsec SAs through Phase 1 & 2 modes of exchange. Additionally, the keys that are used to provide confidentiality and data origin authentication have to be renewed periodically to prevent the compromise of the channel by attackers. However, as the network gets larger the number of SAs needed at each node increases together with the complexity in maintaining all these connections, reserving memory for the SADB and SPD, and renewing keys.

Our goal is the design of a scalable IPsec network that solves these problems in large IPsec networks by constructing a trusted backbone from a given set of nodes that provides security between end-hosts through secure and reliable IPsec gateways. In our scalable architecture, we do not maintain any end-to-end IPsec SAs. Rather, the links that are chosen to be part of constructed network are secured individually using IPsec. Therefore, a chain of IPsec connections or SAs exist between two hosts that want to communicate with one other.

3.1 Objectives

The following are the objectives that we meet through designing the algorithms for the construction of a scalable secure network:

3.1.1 Connected Network

Connectivity is one of the important concepts in networking and graph theory. As Mengers Theorem [27] states, it is the minimum number of edges whose removal results in a disconnected graph. A graph is connected when we have a route from every node to any other node in our graph. In other words, there are no unreachable nodes. The connectivity of a network is a measure of the robustness of the network.

3.1.2 Shortest Path

The shortest path problem is the problem of finding a path between two nodes in such a way that the sum of the weights or costs of the edges belonging to this path is minimized. In our design, the first constraint is to control the cost of the shortest path by bounding the allowed maximum number of SAs required to travel from one host to the other. It is important to have a bound on the cost of a path because long paths cannot achieve fast convergence neither have a low maintenance cost. In addition to that, shortest path reduces the latency and response time of the network.

3.1.3 Degree Constraint

The aim of a degree constrained graph is to have a balance in the distribution of the load in the network. To have a scalable IPsec network, the SAs should be evenly distributed across the nodes in the backbone. The second constraint puts a limit on the maximum number of SAs that are allowed on each node.

3.1.4 Dominating Set

A set of nodes is a dominating set if all the nodes in the network are either in this set or have a neighbor in this set. Applications of a connected dominating set in wireless networks include:

- Reduction in routing overhead: through the removal of links between non-backbone nodes, the maintenance cost and the size of the routing tables can be reduced. We can also avoid excessive broadcast redundancy by having only the backbone nodes forward the broadcast packets.
- Area Coverage: it gives us a good approximation about the node coverage of a connected dominating set.

Although minimizing the size of the connected dominating set will increase the efficiency of the network, it will not take redundancy into consideration. To add redundancy on top of minimizing the size of the Connected Dominating Set (CDS), we need to provide alternate routing paths to the nodes in the network. It is natural for a node to fail due to some damage, energy depletion, and long terms of usage or node movement. The addition of this redundancy will overcome these faults that occur and provide routing flexibility through providing alternate disjoint paths.

3.2 Definitions

Non-Backbone node: if every combination of neighbors of a node has k alternate disjoint paths between each other satisfying degree constraint and path bound,

then the node becomes a non-backbone node. This means that the neighbors can reach one another without this node.

Backbone node: if any combination of neighbors of a node does not have k alternate disjoint paths between each other satisfying degree and path constraints through the backbone, then the node becomes a backbone node. This means that the neighbors cannot reach one another without this node.

Backbone: is a collection of backbone nodes, constructed dynamically, that provides connectivity to the graph. Any node can reach to any other node in the graph through the backbone. The backbone is responsible for carrying information between any two nodes and can tolerate at least $k - 1$ simultaneous node failures while maintaining connectivity.

Cost of an Edge: it represents the number of routers, hops or SAs between two nodes, where the edge exists in between.

Connected Graph: if every node can reach to any other node in the graph, a graph G is said to be connected. In other words, there is a path between any two nodes in the graph.

k -vertex connectivity or k -connectivity: A graph G is k -vertex connected if it is connected and removing $k-1$ nodes from it does not create a disconnected graph of G . In other words, any two nodes in the graph are connected via k -node disjoint paths (Manger's Theorem) [27].

k -CDS: A node set $V' \subseteq V$ is k -dominating set of G if every node is either in the set V' or connected to at least k neighbors in the set V' . A k -dominating set is a k -CDS if the generated subgraph $G[V']$ of V' is k -vertex connected [2].

k -coverage condition: Node v has a non-backbone status if for any two neighbors u and w , k node disjoint alternate paths exist that connect u and w via several intermediate nodes with higher IDs than v [2].

k -constrained-coverage condition: Node v has a non-backbone status if for any two neighbors u and w , k alternate node disjoint paths exist that connect u and w through the backbone via several intermediate nodes with higher priorities than v satisfying the degree and path constraints.

Neighborhood: The neighborhood $N(i)$ of a vertex $i \in V$ is the set of all neighbors of i in G .

3.3 System Design

To obtain a secure link in current networks, a node has to establish an end-to-end secure channel with every other node it wants to communicate with. This has scalability issues in a large network since every host will have SAs equal to two times the number of nodes it wants to establish an end-to-end secure link with. In our design we aim to solve the scalability problem by creating a backbone through Algorithm 1, described in Chapter 4, that contains a subset of these nodes which are selected based on certain parameters (i.e. priority). This allows the non-backbone nodes, also called Non-Backbone IPsec Gateways to establish an IPsec connection with the backbone nodes, also called Backbone IPsec Gateways, they are closest to and that reliable IPsec gateway itself will have secure IPsec connections with a chain of backbone IPsec gateways hence allowing a secure data transmission between any two nodes in the network. It is important to note that all the nodes in the network must be trusted, belonging to the same large network, and hence the ability to create a chain of IPsec SAs.

The nodes that lie outside of the backbone can communicate with any other node in the network through the backbone gateways only, thus having only one SA with the backbone node rather than having an SA with every node in the network. To minimize the number of SAs negotiated inside the backbone, we ensure that any IPsec communication path has minimal cost satisfying the properties of Degree Constrained Shortest Path Tree (DCSPT) and the number of SAs that are part of the path have a certain upper bound that should not be exceeded. It is important to have a limit on the number of SAs, through the degree constraint, that a node can have to solve the problems that traditional IPsec networks suffer from as mentioned above. However, this scalable architecture introduces a very critical problem that traditional IPsec networks did not have. Since we have a chain of SAs that provide security between end-hosts, we have an increase in the latency of the network because of the increase in the number of encryptions and decryptions needed, which is proportional to the number of edges that are part of the IPsec path. Traditional networks did not suffer from this problem since they had one encryption and decryption due to the point-to-point nature of IPsec. Therefore, we untangle this problem by the introduction of the second constraint which is the bound on the number of edges that form a part of the IPsec chain between two securely communicating end-points. Finally to add redundancy against node failures, we use the concept of k -CDS to ensure that every node can reach to any other node in the graph through k alternate disjoint paths. Furthermore, even if one of the nodes is compromised by an attacker, the presence of k alternate paths allows us to consider the compromised node as failing and thus an alternate path with minimal costing chain of IPsec SAs is chosen out of the remaining $k - 1$ possibilities.

As show in Figure 3.1, the backbone IPsec gateways that are responsible for

transmitting the secure data from the non-backbone IPsec gateway, are selected to be part of the backbone because they have a higher priority, they constitute the shortest path from the source towards the destination, they are connected to at least k other backbone nodes, they do not exceed the maximum number of SAs that they are allowed to have and this secure connection satisfies the upper bound on the number of SAs that a path is allowed to contain. This allows the creation of a chain of secure IPsec connections between backbone nodes and between backbone and non-backbone nodes through satisfying the degree and path constraints. The backbone node is responsible for connecting any node with any other node in the graph, whereas a non-backbone node is allowed to communicate only through the backbone nodes it is connected to.

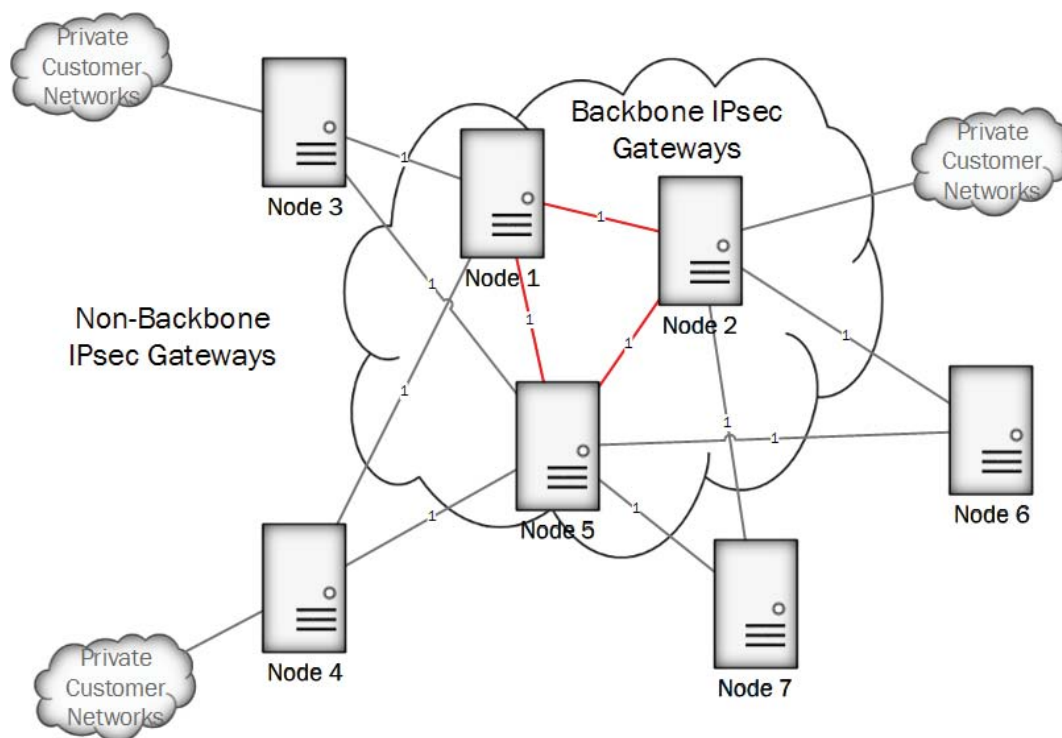


Figure 3.1: Scalable IPsec Network Example

3.3.1 Problem Statement

To formalize our problem and design algorithms to construct our scalable architecture, we used different concepts from graph theory, that are further explained in details in Chapters 4 and 5, and obtained the following problem statement:

Given an undirected k -connected graph $G(V, E)$ with $n=|V|$ representing the

number of nodes in graph G and a cost $c(e)$ associated with every edge $e \in E$ such that $c(e) > 0$. Generate a subgraph G' of G where V' forms a k -CCDS satisfying the following constraints:

- An upper bound m on the cost of a path between nodes i.e. $c(p) = \sum_i^{j-1} c(v_i, v_{i+1}) \leq m$
- A degree constraint vector $\mathbf{d}=(d_v : v \in V)$ i.e. $degree(v) \leq d_v$
- The creation of a k -CDS subgraph with k alternate disjoint paths

3.3.2 Network Architecture

We mapped realistic networks consisting of hosts or end-points such as servers, subnets and virtual machines, and connectors such as Ethernet cables, virtual overlay tunnels and wireless connections into a graph theoretic representation. The network is hence a graph where a host or an end-point is represented as a node, whereas a connector between two hosts is represented as an edge or a link between two nodes. Every edge can have a certain weight or a cost where the cost can represent the bandwidth of a link or the number of intermediate nodes such as routers or switches between its nodes. There are two types of graphs, directed and undirected graphs. A directed graph is a graph where every edge is unidirectional, in other words the traffic can flow from the originating node towards the target node and not the other way around. To support a flow in both directions we need an additional connection from the target node towards the flow originating node. This allows the two nodes to communicate with one another. In an undirected graph the links are bidirectional where a single link is needed to support the flow of data between the two nodes.

In our IPsec architecture, the weights of edges represent the number of SAs or encryption/decryption required between two nodes. Considering the example shown in Figure 3.2, both paths are equivalent to each another. When the weight of an edge is equal to 1 that means that there exists a single encryption and decryption between the two nodes, for example between Node1 and Node2. If we have a weight $w > 1$ that means that there exists $w - 1$ intermediate nodes between end-nodes of the link and hence we have w encryptions and decryptions that are happening along the link, for example between Node2 and intermediate router, and between the router and Node3.

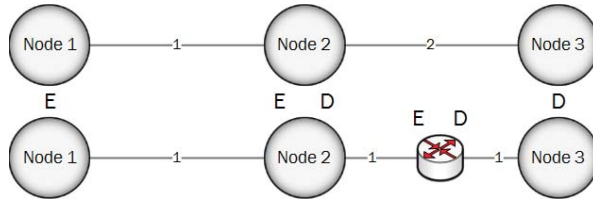


Figure 3.2: Count of Cost of a Path

Thus, the cost of a path will be equal to sum of the costs of the edges and with this consideration we can control the latency originating from these cryptographic functions by bounding them from above by a certain cost m as stated in the problem statement, which in itself is a bound on the number of SAs on a path. In Figure 3.2 for example, the total cost of the path is equal to 3, hence 3 pairs of SAs are needed. Similarly for general network architectures, the weight w of an edge represents the number of hop counts between two nodes. If it is equal to one then there is a direct connection between the nodes, else if it is greater than one that means that there exists $w - 1$ nodes (i.e. routers or switches) between these two nodes.

The links in our architecture are assumed to be undirected edges, and hence the degree of a node is equal to the sum of the links connected to that node. For IPsec networks, the edges are logical overlay links which represent a tunnel between two nodes. As an example consider again Figure 3.2, the degree of Node2 is equal to 2. Although the links are considered to be undirected, the number of SAs used by a node is two times the degree of a node; to provide node-to-node security we need an SA for inbound traffic and another for outbound traffic. To control the number of SAs at each node we introduced the second constraint called the degree constraint vector \mathbf{d} which contains the number of SAs allowed at each node. A degree constraint of two for Node2 means that Node2 is allowed to have a maximum of two connections as part of a shortest path and hence a maximum of two pairs of SAs.

A k -Connected Dominating Set is a set where every node in the network is either in this set or is connected to this set with at least k neighbors. The nodes that are part of the set form the backbone of our network, whereas the nodes that lie outside are the non-backbone nodes that are connected to at least k backbone nodes. Forming a 1-CDS allows us to attain a minimal backbone and thus maximizes efficiency, however it does not provide any redundancy in our network. Therefore to obtain a certain redundancy we use k -CDS and in return minimally increase the size of our backbone. Thus our last constraint, constraint three, states the degree of redundancy of our network. A k -CDS network can tolerate the failure of at least $k - 1$ simultaneous nodes. As an example consider Figure 3.1 from Section 3.3 which is a k -CDS with $k = 2$. We can observe that

every node in our network has at least 2 connections, where the non-backbone nodes have at least 2 connections to backbone nodes. This constraint allows us to maintain connectivity with the failure of at least a node in our network.

In IPsec networks we have an SA between any two nodes as long as the underlying physical network is connected. Since an SA exists between any two nodes, we have a logical overlay full mesh network as can be seen in Figure 3.3(a). This allows us to obtain the following two scenarios:

- When k -CCDS is run on the logical network, we ensure that we have k alternate disjoint paths between any two nodes. However, the underlying physical network might not have this required configuration. This means that when a node fails in the physical network the entire network might get disconnected, but this is not reflected on the logical full mesh network, where when a node fails we still have a $k - 1$ connected graph. However, our aim is to satisfy k -CCDS on the logical network and hence we are not interested in the underlying network because the overlay network is a virtual one.
- The problem that occurs in the first scenario can be solved with the constraint that the underlying physical network is a k connected network. In other words, when a node fails in the logical network it also fails in the physical network but there still exists $k - 1$ alternate disjoint paths in both networks, unlike the first scenario. This allows us to obtain a consistency between the logical and the physical network, bearing in mind that the physical network is not a full mesh whereas the logical network is a logical full mesh, and k -CCDS is still applied on the logical network.

3.3.3 IPsec Gateways

We use the term IPsec gateway because every node in the subnetwork whether it is a non-backbone or backbone node, is connected to private customer networks. The nodes that are part of the private network can communicate with any other node that is also a part of another private network which is connected to a backbone or non-backbone node. Hence, the nodes of our k -Constrained Connected Dominating Set (k -CCDS) network serve as IPsec gateways to the hosts that are part of the private network. Therefore when constructing the architecture of a network, the private networks are not taken into consideration because they connect to other hosts in the network through the IPsec gateways. When a host in a private network wants to communicate with any other host in the network, it creates an IPsec SA with the IPsec gateway it is connected to, which in turn creates a chain of SAs through the backbone nodes to the IPsec gateway of the target host of another private network it wants to communicate with.

One can raise the following question: Shouldn't the SAs created in the private

network be taken into consideration while constructing k -CCDS? Since the nodes in a private network trust one another, all the nodes that are part of the private network can use Group VPN [16] [18]. Group VPN is an extension of IPsec architecture which creates an SA that is shared by a group of devices. Using group VPN, all the hosts that form part of the private network use a single SA created by the key server or group controller to communicate with one another, and the IPsec gateway will serve as the link between the inner and outer networks.

3.3.4 Example of a Scalable IPsec Network

Consider the full mesh represented in Figure 3.3 consisting of five nodes, edge weight 1 or 2, and assume that every node wants to communicate securely with any other node in the network. Since we have an IPsec network and the links are tunnel endpoints, any node can communicate with any other node and hence the need of a full mesh virtual network. The requirements for the generation of the backbone that are inputted to the algorithms for this specific example are: $k=2$, degree constraint $d=3$ and path bound $m = 5$. In traditional networks, Figure 3.3(a), every node has to establish a separate IPsec connection with every other node to have an end-to-end security. Since we have five nodes in our network, every node must have 8 SAs to ensure secure connection with every node and hence a total of 40 SAs is needed. Whereas in the scalable IPsec network, Figure 3.3(b) which was generated by Algorithm 1 described in Chapter 4, Node1 and Node2 were generated to have a non-backbone status whereas the remaining three nodes were chosen as backbone IPsec gateways. The two non-backbone gateways need to be connected to only one of the backbone IPsec gateways to reach securely to any other node in the network but since $k=2$ they should have at least another alternate path for redundancy, and in this particular example they have three possible connections, through nodes Node3 or Node4 or Node5, from which the one with the minimum number of hops and DCSPT satisfaction is chosen. Therefore, nodes Node1 and Node2 require 2 SAs each (one link to connect to the backbone) whereas the remaining contain a minimum of 4 SAs and additional 2 SAs for each non-backbone node if it is connected through a particular backbone node, hence we will obtain a total of 20 SAs. Additionally, this network can sustain the failure of at least a node which will not result in a disconnected network and hence the nodes can communicate with each other as if no failure has occurred, because alternate node disjoint paths exist. For example, assume that node Node3 fails, which results in the network configuration show in Figure 3.3(c). We can observe that the non-backbone IPsec gateways, nodes Node1 and Node2, can still communicate with each other through the path that contains backbone IPsec gateways Node4 and Node5. It is also important to note that in current networks any node has a SA with every other node in the network, whereas in our scalable design the gateways that are not part of the backbone do not have any connection with other non-backbone gateways but rather connect to

every other non-backbone node through the nodes that are part of the backbone.

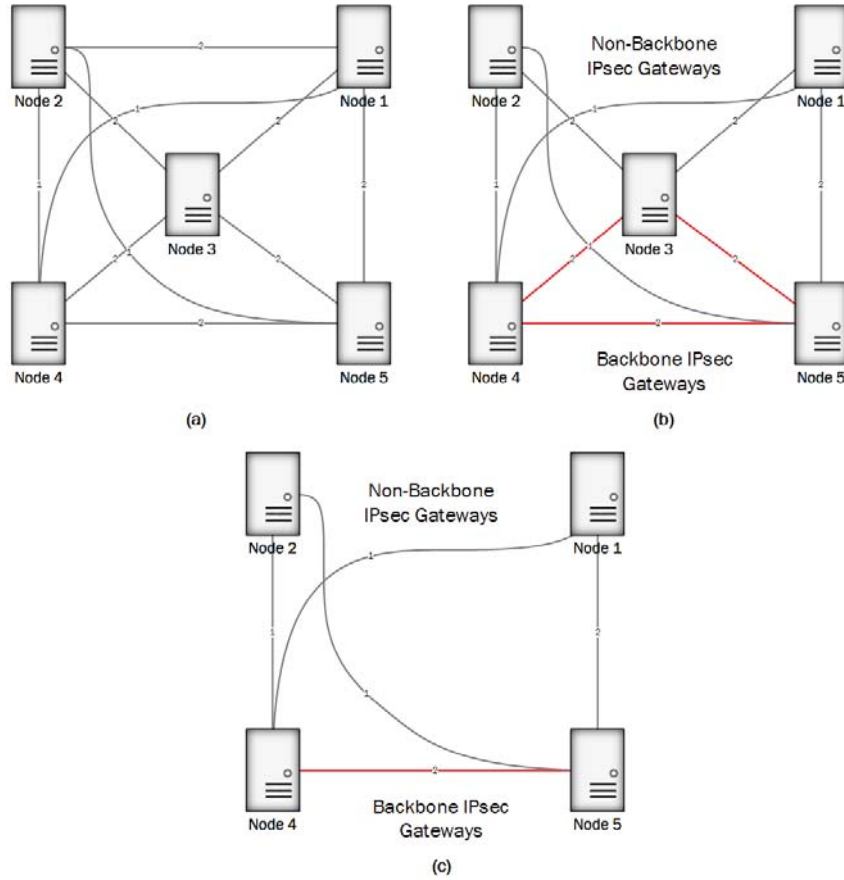


Figure 3.3: Scalable Full-Mesh IPsec Network

3.4 Routing using k -CCDS

A non-backbone node is connected to k other backbone nodes in the network to have alternate paths in the case of failure of up to $k-1$ backbone nodes. Therefore, it can reach to any other node in the network through any of the backbone nodes. Assuming that a node wants to communicate with another node in the network, it has k different possibilities to choose from. The path that will be chosen has to satisfy the degree and path constraints and has to minimize the number of SAs chosen along the path as much as possible. Thus the proposed procedure for non-backbone nodes is to run DCSPT, Algorithm 2 from Chapter 4, from every backbone node it is connected to and choose the path that uses the minimal number of SAs. As for a backbone node, it can directly run DCSPT from itself as the source and choose the returned path to communicate with any

other node in the network which itself contains the minimum number of SAs.

Although the described procedure allows us to route the information correctly and securely by using the shortest path that satisfies the degree and path constraints, the DCSPT algorithm must become an integral part of the routing mechanism of the nodes which leads to the requirement of a new instance of an overlay routing protocol [13]. This requires the modification of the routing protocol of all the nodes that are part of the k -CCDS, which is not feasible, and leads to an increase in the complexity of processing and maintaining these routes with the increase in the size of the network, and the addition of further complexity due to changing routes arising from node failures. An innovative and feasible solution is the utilization of Software Defined Networking (SDN) which is changing the way networks are being designed and managed by introducing a separation between the control and data planes. This separation allows the centralization of network logic and topology in the Network Operating System (NOS) or the control plane, hence allowing the simplification of the network forwarding devices into network switches.

SDN architecture is divided into three different layers with the possibility of virtualization being introduced in any plane. The three layers are: data, control, and application planes. The data plane consists of switches which could either be physical and support the OpenFlow (OF) protocol [28], virtual OF switches such as Open vSwitch (OVS) [29], or a hybrid of both. Data plane switches carry the responsibility of forwarding traffic to correct destinations based on the flow entries in their flow tables which are inserted by the control plane through a southbound API such as OpenFlow. The control plane has an overview of the topology of the entire network and it represents the functionalities of the traditional non-SDN switches being abstracted in the controllers. A controller translates the requirements and policies from the application plane, obtained through a northbound API such as REST, to OF rules and enforces them on the data plane switches. The application plane is a new virtualization layer introduced through SDN, which allows developers to write applications and convey their requirements to the corresponding controllers.

The Open vSwitch Database (OVSDB) management protocol [30] [31] is an OpenFlow configuration protocol that is designed to control and manage data plane switch implementations, particularly OVS implementation. OVSDB consists of the OVSDB-server and a switch daemon that configures the OVS switches based on the configuration information that it obtains from the controller through the OVSDB-server. The controller is connected to every node in the network, hence it has an overview of the topology of the network and the ability to configure them using the OVSDB management protocol. k -CCDS can be run as an application on the controller, which will be used to identify the backbone and non-backbone nodes or switches and accordingly, based on the marking process, the links that will be protected through IPsec. Using OVSDB, the marked links

that are part of the backbone will be configured to be IPsec tunnels and every non-backbone will be configured to have a tunnel with a backbone node. As for the routing problem that has been mentioned, the application running on the controller will determine for each node through DCSPT the path, out of k possibilities, that is the shortest satisfying the constraints and hence the SAs or tunnels that will be used to reach to all other nodes in the network. After determining the IPsec links that will be used, the controller can easily install the corresponding flow rules on all the nodes of the network and thus allow routing of the packets securely between any two nodes by using the shortest path satisfying the degree and path constraint. Since the controller and switches exchange OF EchoRequest and EchoReply messages for liveness, bandwidth and latency information, when a backbone node fails in the network the controller will not receive any of these messages from the failing node and thus will conclude that this node is not part of the network anymore. The non-backbone nodes that used to connect to the network using the failing backbone node will not be connected anymore, but since through DCSPT the controller knows the remaining $k-1$ alternate disjoint paths for this non-backbone node, it will determine the next shortest path and install the corresponding flow rules on the network nodes and thus maintain the connectivity in the network.

3.5 k -CCDS Security vs. Scalability

As we have seen in Chapter 2, there is always a trade-off between security and scalability. In our system we do not have any point to point IPsec tunnels, rather a chain of IPsec SAs which allows us to reduce the number of SAs needed to obtain a secure and scalable connected network. When a backbone node obtains a packet that should be forwarded to the IPsec chain's next backbone node, it firstly decrypts the packet because it has a direct tunnel with the first backbone node that it is obtaining the packet from, encrypt the packet again based on the SA and SP entries for the IPsec connection with the chain's next backbone node, and finally forward the encrypted packet to eventually reach securely to the destination. Since a node is decrypting and encrypting the packets, during the interval of after-decryption and before-encryption the packet's content will not be confidential and thus can be read by the node. But since we have assumed during our design that the network could be trusted as in MPLS VPNs, this would not be problematic and thus allow us to preserve networks security.

However, during the event of exploitation of the node by an attacker, we have the following two scenarios; if we know that the node is under the control of the attacker, what we can do is assume that the node has failed and thus choose an alternate path out of the remaining $k-1$ possibilities. Therefore, this node will not be used by the network anymore and will be considered to be nonexistent

which solves the security problem. However if we cannot detect that the node is under the control of a malicious user, then only the paths that go through this node will be exploited because now the attacker can read all the packet content during that small window of time. We have the same problem in Group VPN and Get VPN; if the group controller or any group member is exploited not only part of the group is affected, rather the whole group is exploited because every group member can encrypt and decrypt the packets of any other other member through sharing a group SA and thus the attacker can decrypt all the packets. Although k -CCDS does not solve the latter problem, however it allows the user to choose a middle-ground solution. The algorithm allows the constructed network to range from a very scalable and weakly secured network eg. the star topology to a non-scalable and highly secured network eg. full-mesh topology. The constraints of k -CCDS provides the user with the flexibility of obtaining a network based on the priority of the requirements eg. scalability, security, redundancy, latency etc. To elaborate the idea further, assume that a network consists of 1000 nodes and a user wants to create a scalable and secure IPsec network. If very strong security and redundancy is not a top priority for the user, eg. traffic contains video and audio streaming, the user can choose a very scalable design by setting $d=999$, $m=2$ and $k=1$ which generates the star topology and reduces the number of SAs from 2,000,000 to 3996. If very strong security is a top priority eg. traffic containing private information, then scalability cannot be provided because of the trade-off between them, the user can choose $d=999$, $m=1$, and $k=2$ which generates the full mesh topology and thus obtain 2,000,00 SAs, which prevents the effect of exploit of a node on any other node in the network. If a user requires certain scalability and high security, then he can choose $d=50$, $m=3$ and $k=2$ which generates a k -CCDS subgraph where the number of SAs are reduced from 2,000,000 to 4008, and thus construct a secure and scalable network. As observed, the algorithm gives the flexibility to the user to choose either scalability, security, or a mixture of both. As for the second security problem that we have mentioned, if only a specific application requires a very high security, SSL is a solution over the k -CCDS network for that particular application which provides end-to-end security for the application specific packets. Furthermore, Group and GET VPN provide scalability by extending the IPsec protocol, however k -CCDS uses IPsec without any modification to construct scalable secure networks.

Chapter 4

Algorithms

In this chapter we demonstrate and explain the different algorithms that we have used to construct the scalable architecture we have described in Chapter 3.

The aim of the first algorithm is to construct a subgraph of G , k -Constrained Connected Dominating Set (k -CCDS), consisting of backbone and non-backbone nodes that form a k Connected Dominating Set (k -CDS) satisfying the degree and path constraints. The foundation of the algorithm is based on the research done in [2] where the authors introduced the k -Coverage condition. k -CCDS starts initially by sorting the nodes in descending order of priority, where the priority of a node is based on the degree of a node and/or unique ID where the nodes that have a higher degree and/or larger ID have a higher priority. The notion of priority is important because first, it allows us to know which nodes will be part of the backbone and second, it avoids the destruction of the connectedness of the graph when $k-1$ nodes fail simultaneously. This allows us to create the backbone dynamically because when a node is tested for a combination of neighbors reaching one another, the path has to go through the “current” backbone. Therefore, the node with the highest priority will always be part of the backbone. As the number of nodes in the backbone increases, the probability of a node being non-backbone node increases because it has more chance of finding k alternate disjoint paths satisfying the degree and path constraints, through the backbone, for its combination of neighbors. Initially, all the nodes are marked as non-backbone nodes and we start from the highest priority node. For a node to remain a non-backbone node, every combination of its neighbors are checked if they have k alternate disjoint paths, through the backbone, between each other that satisfies the degree and path constraints (lines 3-29). If every combination of neighbors has k alternate paths satisfying both constraints, then the node remains a non-backbone node. However, if there exists at least one combination of neighbors that does not satisfy any of the three stated conditions, then the node is marked to be a backbone node, thus expanding the backbone by this node (lines 25-32). A path between two neighbors of a node is found by running

DCSPT, Algorithm 2, which returns us Dijkstra’s shortest path, if it exists, where every node satisfies the given degree constraint (line 13). When no DCSPT is returned, that means there exists at least one node on the path that cannot satisfy the given degree constraint \mathbf{d} (lines 14-16). If Algorithm 2 returns a DCSPT, then that means the degree constraint of every node of the path is satisfied. We proceed by calculating the number of hops between these two neighbors which is the sum of the costs of the edges of the path returned by DCSPT (line 17). If the calculated distance does not exceed the path bound, then the path bound constraint has been satisfied. This means that we have found 1 disjoint path satisfying both constraints. What remains to satisfy is $k - 1$ such alternate paths (lines 18-19). To check if the last constraint is satisfied, we temporarily remove the nodes that are part of the newly found path from G (line 20), and repeat the same procedure again (finding DCSPT and checking path bound). This allows us to know if k alternate disjoint paths exist between these neighbors that satisfy the degree and path constraints. This whole procedure is repeated for every two neighbors of every node of G to construct our backbone.

Given that a solution exists, i.e. the inputs k , path bound and degree constraint can find the desired subgraph, the algorithm will construct a k -CCDS which tolerates the failure of at least $k-1$ nodes. If a solution does not exist, then the last part of the algorithm will find a solution by relaxing the degree and path constraints (lines 34-52). The algorithm starts by assuming that no node is failing in the graph, and hence for each backbone node it runs DCSPT to check if it can reach to every node in the graph by satisfying the degree constraint. If there exists at least one node on the path that does not satisfy the degree constraint, DCSPT returns the violating nodes. What the algorithm does is that it fixes the degree constraint of that violating node by incrementing it by 1 (lines 39-43). This procedure is repeated until no node on the path is violating the degree constraint. This new degree constraint is called the relaxed degree constraint. After obtaining the relaxed degree constraint, we find the longest path of this shortest path, diameter of the path, by running Dijkstra’s shortest path algorithm from the node that is furthest from the evaluated backbone node. Hence, we obtain the cost of the diameter. We compare it with the path bound and if it is greater, then we relax the path bound constraint by setting the path bound equal to the cost of the diameter of the tree (lines 45-49). The same procedure is repeated by removing all possible combinations of $1, 2, \dots, k - 1$ nodes from the graph, testing if the constraints are not satisfied and if not satisfied relaxing the constraints to find a solution. Therefore, we can find a k -CCDS that satisfies the relaxed conditions. The relaxed conditions can further be used to run Algorithm 1 again to obtain a k -CCDS with a smaller backbone. Finally, it is important to note that we mark the edges connecting backbone nodes that haven’t been used (are not part of any DCSPT) to remove them from our network architecture (line 45). Let n_b represent the number of backbone nodes, then the running time of k -CCDS is in the order of $O(kdn_b n^3 2^n + k \binom{n_b}{k} d n_b^2 2^n)$.

Algorithm 1 k -CCDS(G, \mathbf{d}, k, m)

Input: Graph $G (V, E)$, degree constraint vector $\mathbf{d} = (d_v : v \in V)$, the number of alternate paths k and the upper bound on path cost m

Output: k -CCDS subgraph of $G (V, E)$ satisfying degree and path constraints which contains the Backbone Status of nodes, fixed/relaxed \mathbf{d} and m if they exist.

- 1: **Initialize** the status of each node in G as a non-backbone node.
- 2: Assign a unique priority to every node in G by sorting the degrees in descending order and/or assigning a larger unique ID for higher priority nodes.
- 3: **for** every node $v \in G[V]$ in descending priorities **do**
- 4: $nonBackboneStatus = 1$
- 5: **for** every combination of neighbors u and w of v **do**
- 6: $l = 0$ // l represents the number of disjoint paths obtained from u
- 7: Temporarily remove node v from graph G
- 8: Temporarily decrement d_u by 1: $d_u = d_u - 1$
- 9: **for** the size of $degree(u)$ **do**
- 10: **if** $l == k$ **then**
- 11: **break** // We found k alternate disjoint paths
- 12: **end if**
- 13: Call **DCSPT**($G, \mathbf{d}, u, \text{nodes with Backbone status}$)
- 14: **if** **DCSPT** cannot be found **then**
- 15: **break**
- 16: **end if**
- 17: $numberOfHops = \text{DCSPTdistance}(u, w)$
- 18: **if** $numberOfHops \leq m$ **and** u and w reach one another **then**
- 19: $l = l + 1$ // We have found a node disjoint path
- 20: Disregard the nodes that were part of this path to find a new disjoint path during the next iteration
- 21: **else**
- 22: **break**
- 23: **end if**
- 24: **end for**
- 25: **if** $l < k$ **then**
- 26: $nonBackboneStatus = 0$
- 27: **break** // if any combination fails, this node cannot be a non-backbone
- 28: **end if**
- 29: **end for**
- 30: **if** $nonBackboneStatus == 0$ **then**
- 31: Change status of v to backbone
- 32: **end if**
- 33: **end for**
- 34: // If $0, 1, \dots, k-1$ nodes fail, check whether the constraints are still satisfied. If they are not satisfied, we relax them.

```

35: for  $r=0,1,\dots,k-1$  do
36:   for every combination of  $r$  backbone nodes do
37:     Temporarily remove the combination of  $r$  backbone nodes when  $r > 0$ 
38:     for every remaining node  $v$  that has backbone status do
39:       repeat
40:         Call DCSPT( $G, \mathbf{d}, v, \text{nodes with Backbone status}$ )
41:         if DCSPT returns violating node then
42:            $d_{violating} = d_{violating} + 1$ 
43:         end if
44:       until no violating node is returned
45:       Mark the edges that are part of DCSPT
46:       Run Dijkstra from node  $u$  that is furthest from  $v$  to obtain the longest shortest
47:       path  $P_{sl}$  of the tree
48:       if  $P_{sl} > m$  then
49:          $m = P_{sl}$ 
50:       end if
51:     end for
52:   end for

```

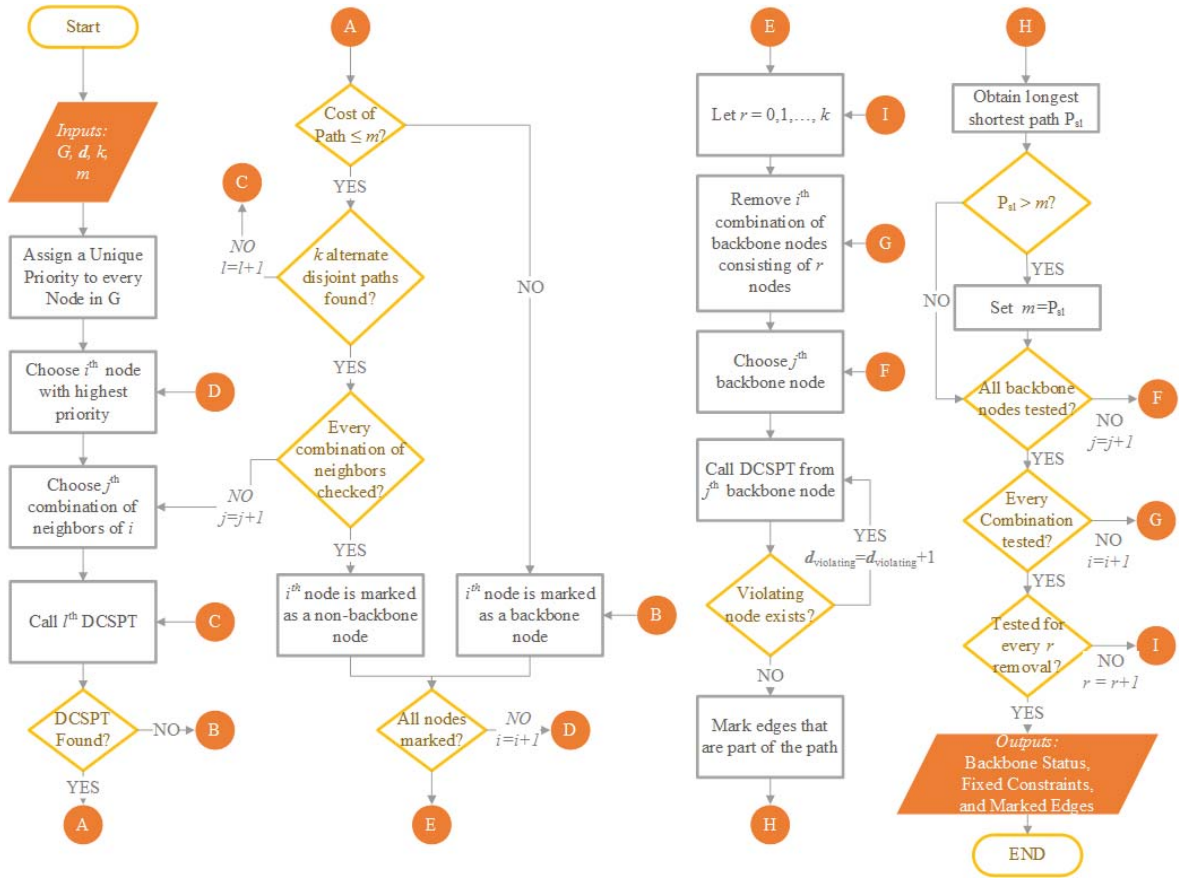


Figure 4.1: Flow Chart of Algorithm 1

Algorithm 2, Degree Constrained Shortest Path Tree, aims at finding a degree constrained shortest path by fixing the nodes, found on Dijkstra's path, violating the provided degree constraint. This algorithm is a variation of the idea that has been provided by the authors in [24] [25] [19]. It initially calls Constrained Dijkstra's algorithm, Algorithm 3, that returns a shortest path that runs through the backbone (line 1). In other words, non-backbone nodes are not allowed to be the link between backbone nodes or between a backbone or non-backbone node. The next step is to check for every backbone node starting from the top of the height of the tree if they are violating the degree constraint (lines 2-4). We start from the top of the height of the tree because once a node is fixed and we go further down the tree, even if changes are done in the lower part, the upper part will not be affected because nodes that are further down the tree are descendants of the nodes of the upper level of the tree, and hence any change in the lower parts effects on the descendants that are in further lower levels of the tree and not on the nodes, parents, that are at higher levels of the tree. Non-backbone nodes are not checked because constrained Dijkstra allows them to have only

one connection and that is only with a backbone node. If there does not exist a node violating the given degree constraint, the result is returned without any modification. However, if a backbone node is violating the degree constraint we need to fix it, in return increasing the cost of the shortest path with a minimal penalty. Hence, we have a tradeoff between fixing violating nodes and obtaining a minimal shortest path.

When fixing a node v violating the degree constraint, every child c of the violating node is checked if it can be connected to any other node p in the graph, where the edge in between is not part of Dijkstra's shortest path generated by Algorithm 3 and the new node p can accept new connections without violating its degree constraint. Furthermore it cannot be the case that the child node c is a backbone node and the penalty node p is a non-backbone node, or c and p are non-backbone nodes (line 7). We have this condition because we cannot allow non-backbone nodes to have connections with each other since they are allowed to connect to others only through the backbone, and we cannot allow backbone nodes to be connected to the path through a non-backbone node. For every child node c that can be connected to other nodes p through satisfying the stated conditions, the node that adds a minimal penalty on the cost of the shortest path will be chosen based on the following formula (lines 8-11):

$$Penalty_c = \min_{\forall p} [distance(s, p) + cost(c, p) - distance(c, c)] \times (numberOfDescendants(c) + 1)$$

Once the minimal penalty node is obtained for each child node, the minimal one of all the children is chosen to be disconnected from the violating node and to be reconnected to the new penalty node (lines 13-16):

$$AddedPenalty = \min_{\forall c} Penalty_c$$

After reconnecting to the new penalty node, we need to increase the distance of the descendants and the child node from the source by $distance(s, m) + cost(m, u) - distance(s, u)$ (line 15). By continuing this process, Algorithm 4 is called which checks for each descendant of the child node now connected to the penalty node, if it can be reconnected to other node to reduce the penalty, hence minimizing the total cost of the shortest path (lines 17-19). The process is repeated until no backbone node is violating the degree constraint. This procedure allows us to obtain a DCSPT. Let d represent the maximum number of times the degree of a violating node has been fixed and n_b the number of backbone nodes, then the running time of Algorithm 2 is in the order of $O(dn_b2^n)$

Algorithm 2 DCSPT($G, \mathbf{d}, s, \mathbf{B}$)

Input: Graph $G (V, E)$, degree constraint vector $\mathbf{d} = (d_v : v \in V)$, source s and nodes that have backbone status \mathbf{B} .

Output: Degree Constrained Shortest Path Tree (DCSPT) satisfying \mathbf{d} or an empty set and the violating node if DCSPT cannot be formed.

```
1: Form SPT from  $s$  using modified Dijkstra's single source shortest path algorithm constrainedDijkstra ( $G, s, \mathbf{B}$ )
2: for each level  $p = 0$  to  $height(\text{SPT})$  do
3:   for each node  $v$  of level  $p$  that has backbone status in  $\mathbf{B}$  do
4:     while  $degree(v) > d_v$  do
5:       for every child  $u$  of  $v$  do
6:         for each node  $m$  that is connected to  $u$  but edge  $\notin$  SPT do
7:           if  $degree(m) < d_m$  and  $m$  is not a descendant of  $u$  and
               $\neg (\mathbf{B}(u) == 0 \text{ and } \mathbf{B}(m) == 0)$  and
               $\neg (\mathbf{B}(u) == 1 \text{ and } \mathbf{B}(m) == 0)$  then
8:              $penalty_u(m) = [\text{distance}(s, m) + \text{cost}(m, u) - \text{distance}(s, u)]$ 
               $\times (\text{numberOfDescendants}(u) + 1)$ 
9:           end if
10:        end for
11:       Choose the minimal penalty  $penalty_u$  of node  $u$  among all  $m$ 
12:     end for
13:     Choose  $u$  that has the minimal  $penalty$  among all children of  $v$ .
14:     Disconnect  $u$  from  $v$ 
15:     Update the distance between  $u$  and the nodes that are its descendants by
               $[\text{distance}(s, m) + \text{cost}(m, u) - \text{distance}(s, u)]$ 
16:     Connect  $u$  to  $m$ 
17:     if  $u$  has descendants then
18:       UpdateDescendants( $G, u, \mathbf{B}$ )
19:     end if
20:   end while
21: end for
22: end for
```

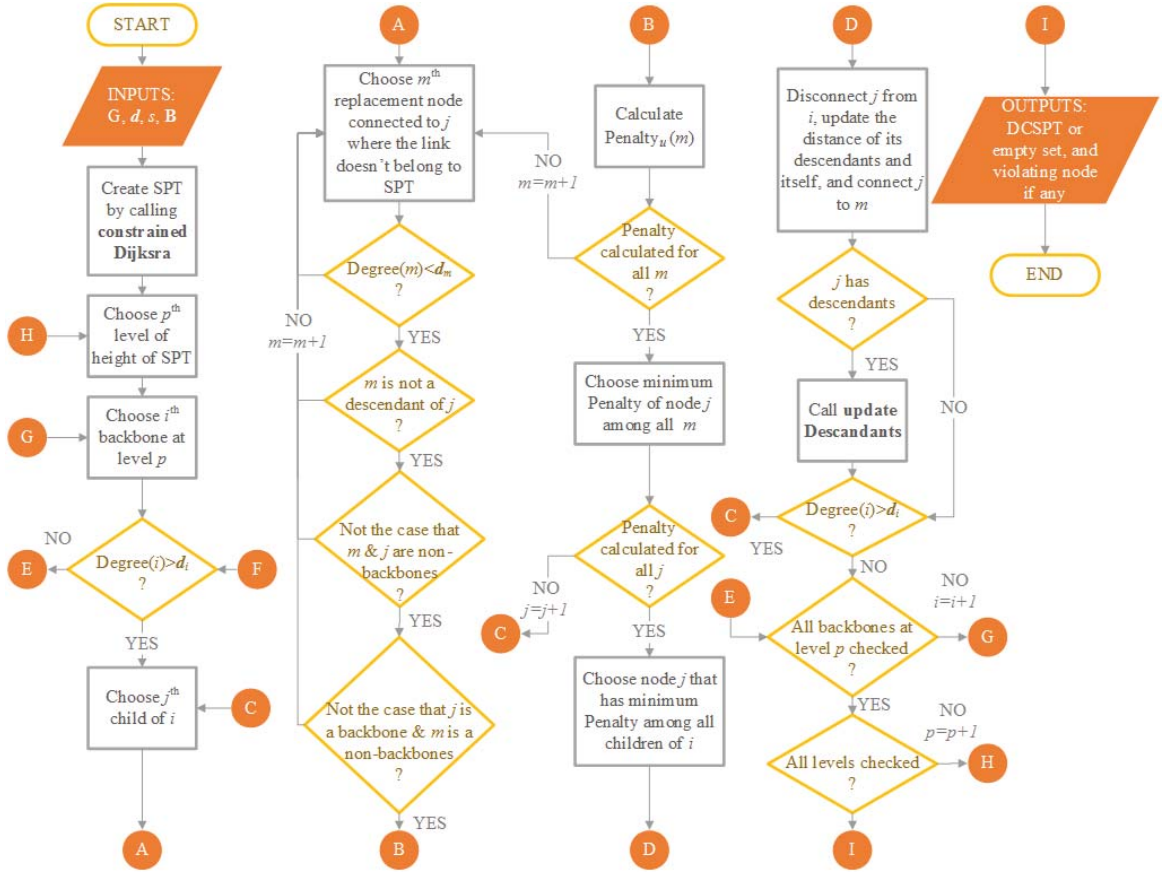


Figure 4.2: Flow Chart of Algorithm 2

Algorithm 3, Constrained Dijkstra, is similar to Dijkstra's algorithm, however it does not allow any nodes that are not part of the backbone yet to be the link between any two nodes. In other words, a non-backbone node cannot be the link that connects a backbone or non-backbone node to any other node in the graph. Hence, only backbone nodes can be the link between any two nodes. The remaining nodes that are not part of the backbone, so far, will have a single connection to a backbone node depending on Dijkstra's algorithm. As for the variable i that has been introduced, whenever Dijkstra's source s is a non-backbone node, then initially we'll allow it to reach to one backbone node only and not to any other backbone or non-backbone node. As for the remaining non-backbone node, since i is incremented, Constrained Dijkstra will not allow it to form a link between any nodes and thus it will connect them to backbone nodes only (lines 13-22).

Algorithm 3 ConstrainedDijkstra(G, s, \mathbf{B})

Input: Graph $G (V, E)$, source s and nodes that have backbone status \mathbf{B}

Output: Constrained Dijkstra's shortest path tree.

```
1: Create vertex set Q
2: for each node  $v$  in graph  $G$  do
3:   distance( $v$ )= $\infty$ 
4:   parent( $v$ )=NULL
5:   Add  $v$  to Q
6: end for
7: distance( $s$ )=0
8:  $i = 0$ 
9: while  $Q \neq \emptyset$  do
10:   $u$ =node in  $Q$  with minimum distance
11:  Remove  $u$  from  $Q$ 
12:  for each neighbor  $v$  of  $u$  do
13:    if ( $\mathbf{B}(v)==0$  and  $\mathbf{B}(u)==0$ ) or ( $\mathbf{B}(v)==1$  and  $\mathbf{B}(u)==0$  and  $i \neg = 0$ )
14:      then
15:        continue
16:      end if
17:       $altDistance$ =distance( $u$ )+cost( $u, v$ )
18:      if  $altDistance < distance(v)$  then
19:        distance( $v$ )= $altDistance$ 
20:        parent( $v$ )= $u$ 
21:         $i = i + 1$ 
22:      end if
23:    end for
24:  end while
```

Update Descendants, Algorithm 4, is the continuation of Algorithm 2 where we check if we can minimize the penalty that has been added to the shortest path. For every descendant node of a child node that has been connected to a new node in Algorithm 2, we check if we can reconnect the descendants to other nodes in the graph and minimize the added penalty to the shortest path. Every descendant r of the child u is checked if it can be connected to any other node f in the graph, where the edge in between is not part of Dijkstra's shortest path generated by Algorithm 3 and the new node f can accept new connections without violating its degree constraint. Furthermore, it cannot be the case that r is a backbone node and the penalty node f is a non-backbone node, or r and f are non-backbone nodes (line 4). Additionally, we calculate the amount with which the penalty can be reduced and check if this penalty is negative:

$$Penalty_{r,f} = \min_{\forall f} [\text{distance}(s, f) + \text{cost}(f, r) - \text{distance}(s, r)]$$

For every penalty minimizing node f that can be connected to descendant node r through satisfying the stated conditions and whose added cost is negative, the node that adds a minimal penalty, largest negative, on the cost of the shortest path will be chosen. Hence, the descendant node r is disconnected from u and is now connected to the new penalty minimizing node f (lines 8-10). Furthermore, the distance of the descendants from the source node is updated by adding $\text{distance}(s,f)+\text{cost}(f,r)-\text{distance}(s,r)$ to it, which is a negative value (line 11). This procedure allows us to minimize the penalty that has been added to the DCSPT generated by Algorithm 2. Now, since the descendants of r are not part of the descendants of u , this algorithm will not be applied on the descendants of r anymore, hence we can run this algorithm recursively on the descendants of r by calling $\text{UpdateDescendants}(G,r,\mathbf{B})$.

Algorithm 4 $\text{UpdateDescendants}(G,u,\mathbf{B})$

Input: Graph $G(V, E)$, node u and nodes that have backbone status \mathbf{B}

Output: DCSPT with minimized penalty

- 1: **for** every descendant r of u **do**
 - 2: **for** every node f that is connected to r but edge \notin SPT **do**
 - 3: Calculate $\text{penalty}=\text{distance}(s,f)+\text{cost}(f,r) - \text{distance}(s,r)$
 - 4: **if** $\text{degree}(f)< d_f$ **and** f is not a descendant of r **and**
 $\neg (\mathbf{B}(r)==0 \text{ and } \mathbf{B}(f)==0)$ **and**
 $\neg (\mathbf{B}(r)==1 \text{ and } \mathbf{B}(f)==0)$ **and** $\text{penalty}< 0$ **then**
 - 5: Make this node f a possible candidate of exchange
 - 6: **end if**
 - 7: **end for**
 - 8: Choose the minimal penalty of node r among all f
 - 9: Disconnect r from its parent u
 - 10: Connect r to f
 - 11: After modifying the connection, update the distance of r from s and the nodes that are descendants of r by $\text{distance}(s,f)+\text{cost}(f,r)-\text{distance}(s,r)$
 - 12: [Optionally we can call $\text{UpdateDescendants}(G,r,\mathbf{B})$ recursively to further minimize the penalty]
 - 13: **end for**
-

Chapter 5

Proof of Algorithms

5.1 Problem Statement

Given an undirected k -connected graph $G(V, E)$ with $n=|V|$ representing the number of nodes in graph G and a cost $c(e)$ associated with every edge $e \in E$ such that $c(e) > 0$. Generate a subgraph G' of G where V' forms a k -CCDS satisfying the following constraints:

- An upper bound m on the cost of a path between nodes i.e. $c(p) = \sum_i^{j-1} c(v_i, v_{i+1}) \leq m$
- A degree constraint vector $\mathbf{d}=(d_v : v \in V)$ i.e. $degree(v) \leq d_v$
- The creation of a k -CDS subgraph with k alternate disjoint paths

5.2 Proof of Theorems

Assuming that a solution exists where the path bound and degree constraint satisfy k -CCDS construction, we can thus prove the correctness of our algorithms by proving the following:

Theorem 1: A non-backbone node u has at least k connections to the backbone, or k alternate disjoint paths exist between every combinations of neighbors of u .

Proof. Assume that a non-backbone node u does not have k connections to the backbone or k alternate disjoint paths do not exist between at least a combination of neighbors of u .

In Algorithm 1, DCSPT is called for every combination of neighbors of node u , $\forall v_i, v_j \in N(u)$ where $i \neq j$, which calculates the degree constrained shortest

path from v_i to v_j based on Algorithm 2. Algorithm 2 initially runs Algorithm 3, Constrained Dijkstra, and proceeds by fixing the nodes that violate the degree constraint. If there still $\exists v \in (v_i, v_{i+1}, \dots, v_j)$ where $degree(v) > d_v$, Algorithm 2 returns an empty set ϕ .

If an empty set ϕ is returned for any combination of neighbors, u is marked as a backbone node. If a non-empty set is returned which means that $degree(v) \leq d_v \forall v \in (v_i, v_{i+1}, \dots, v_j)$, but if the other neighbor v_j is not reachable, an infinite distance between them by Dijkstra's algorithm, the node is marked as a backbone node.

Given that ϕ is not returned and all nodes are reachable, DCSPT(v_i) is run at most k times $\forall v_i, v_j \in N(u)$ where $i \neq j$, and if during any iteration a degree constrained shortest path has $c(v_i, v_{i+1}, \dots, v_j) > m$, the node is marked as a backbone node. However, if k alternate disjoint paths satisfying $c(v_i, v_{i+1}, \dots, v_j) \leq m$ and $degree(v) \leq d_v \forall v \in (v_i, v_{i+1}, \dots, v_j)$ are found through the backbone for every combination of neighbors v_i and v_j of node u where $i \neq j$, then node u is marked as a non-backbone node contradicting to the assumption that a non-backbone node does not have k connections to the backbone. Therefore, a non-backbone node has at least k connections to the backbone of the subgraph G' . ■

Theorem 2: The k alternate disjoint paths of every combination of neighbors of a non-backbone node satisfy the degree constraint.

Proof. Assume that k alternate disjoint paths of at least a combination of neighbors of a non-backbone node u do not satisfy the degree constraint. Then $\exists v_i, v_j \in N(u)$ where $i \neq j$ for at least one out of k alternate disjoint paths that does not satisfy the degree constraint. This means that, Algorithm 2 returns an empty set because $\exists v \in (v_i, v_{i+1}, \dots, v_j)$ where $degree(v) > d_v$ which cannot be fixed in any possible way. Hence Algorithm 1 will assign this node as a backbone node contradicting the assumption that this node is a non-backbone node. Therefore, the k alternate disjoint paths of any combination of neighbors of a non-backbone node satisfy the degree constraint. ■

Theorem 3: Every combination of neighbors of a non-backbone node can reach one another through k alternate disjoint paths satisfying the degree constraint.

Proof. Assume that there exists a combination of neighbors of a non-backbone node u that cannot reach one another through k alternate disjoint paths satisfying the degree constraint. Then $\exists v_i, v_j \in N(v)$ where $i \neq j$ of a non-backbone node u that reach to one another with less than k alternate disjoint paths satisfying the degree constraint. Hence, there exists some alternate disjoint paths that do not satisfy the degree constraint. Thus by proof 2, this node should be a backbone node contradicting to the assumption that it is a non-backbone node. Therefore, every combination of neighbors of a non-backbone node can reach one another through k alternate disjoint paths satisfying the degree constraint. ■

Theorem 4: Given that a path exists through the backbone and the degree constraint of all nodes is satisfied, Algorithm 2 (DCSPT) finds a minimal cost Dijkstra path.

Proof. Given that a path exists through the backbone and the degree constraint of all nodes is satisfied, Algorithm 2 (DCSPT) finds a non-minimal cost Dijkstra path.

Algorithm 2 initially calls Algorithm 3 which is constrained Dijkstra. Algorithm 3 returns the shortest path $P_s = (v_i, v_{i+1}, \dots, v_j)$ from a source node s , where certain nodes could be violating the degree constraint $degree(v) > d_v$ for $v \in P_s$. Algorithm 2 tries to fix the degree constraint of the nodes that are violating it by breaking certain connections and reconnecting them to some other nodes that do not violate the degree constraint, but in return increases the shortest path with a certain penalty. Given that the degree constraint of every node is satisfied, then a minimal cost Dijkstra path exists that connects the nodes together.

If $\exists v \in P_s$ where $degree(v) > d_v$ and it cannot be fixed, then a solution does not exist and hence the algorithm returns an empty set. But we initially assumed that $degree(v) \leq d_v \forall v \in V$, hence this could not be the case and the degree constraint of all violating nodes will be fixed.

When fixing a node v violating the degree constraint, every child c of the violating node is checked if it can be connected to any other node p in the graph, where $(c, p) \notin P_s$ and $degree(p) < d_p$. For every child node c that can be connected to other nodes p , the node that adds a minimal penalty on the cost of the shortest path will be chosen based on the following formula:

$$Penalty_c = \min_{\forall p} [distance(s, p) + cost(c, p) - distance(c, c)] \times (numberOfDescendants(c) + 1)$$

Once the minimal penalty node is obtained for each child node, the minimal one of all the children is chosen to be disconnected from the violating node and to be reconnected to the new penalty node. The process is repeated until no node is violating the degree constraint:

$$AddedPenalty = \min_{\forall c} Penalty_c$$

Since we are always choosing the minimal penalty out of all possibilities, then the cost that is added to the original path generated by Algorithm 2 will be minimal. Furthermore, Algorithm 4 is used to further minimize the penalties, due to fixing the degree of violating nodes in DCSPT, by reconnecting the descendants of the child to other nodes that minimize the added penalty by adding the following negative value:

$$Penalty_r = \min_{\forall f} [distance(s, f) + cost(f, r) - distance(s, r)]$$

Therefore, the minimal cost shortest Dijkstra path satisfying the degree constraint is obtained, contradicting the assumption that Algorithm 2 (DCSPT)

finds a non-minimal cost Dijkstra path. ■

Theorem 5: If the k -constrained-coverage condition is applied to a k connected network G , the resultant backbone V' forms a k -CDS of G satisfying the degree and path constraints. The resultant backbone V' is said to form a k -CCDS of G .

Proof. First we recall the following Lemma and Theorem that have been proven in [2]:

Lemma: A node set V' is a k -CDS of network G if after removing any $k - 1$ nodes from V' , the remaining part of V' is a CDS of the remaining part of G .

Theorem: If the k -coverage condition is applied to a k connected network G , the resultant virtual backbone V' forms a k -CDS of G .

Therefore, Theorem 5 is a corollary of the theorem that has been provided by the authors in [2]. Assume that the resulting backbone V' does not form a k -CCDS of G . Then the generated k -CCDS does not satisfy the degree and path constraints. However, we have assumed that a solution exists and based on the provided lemma, theorem and k -constrained-coverage definition, we have a contradiction i.e. the given degree and path constraints satisfy k -constrained-coverage condition. Therefore, if the k -constrained-coverage condition is applied to a k connected network G , the resultant backbone V' forms a k -CCDS of G . ■

Theorem 6: If a given set of conditions (degree and path constraints) cannot be satisfied in the backbone, Algorithm 1 finds the relaxed conditions.

Proof. Assume that Algorithm 1 does not find the relaxed conditions. Then there exists some node in the backbone that returns an empty set ϕ when Algorithm 2 is run, or when Dijkstra path P_s is returned we obtain $c(P_s) > m$.

The last part of Algorithm 1 checks for every backbone node if they satisfy the constraints when $0, 1, \dots, k - 1$ combination of backbone nodes are removed from the backbone V' . For every remaining backbone node, we run Algorithm 2 from it. If an empty set is returned, then the violating node is also returned and hence the violator's degree constraint is incremented, $d_{v_{violating}} = d_{v_{violating}} + 1$ and Algorithm 2 is run again. The process continues until $degree(v) \leq d_v \forall v \in P_s = (v_i, v_{i+1}, \dots, v_j)$ and hence a non-empty set is returned. Similarly, the path constraint is checked. The longest shortest path for that Dijkstra path P_s is found as we have shown in proof 4. If it does not satisfy the path bound $c(P_s) > m$, then the path bound is set to the value of this path, $m = c(P_s)$. The same process continues for every backbone node. Eventually, the path bound will be equal to the diameter of the graph; hence all paths will satisfy the path bound. Similarly, the degree constraints of all the nodes will be satisfied, contradicting the assumption that Algorithm 1 does not find the relaxed conditions. Therefore, Algorithm 1 finds the relaxed conditions. ■

Theorem 7: Given a large full mesh network where edges are equally weighted, and k -CCDS exists (no need to relax conditions). If the first non-backbone neighbor of the “current” evaluated node reaches to any other non-backbone neighbor through the backbone satisfying the three constraints, then this evaluated node is a non-backbone and the backbone has been found.

Proof. Based on Algorithm 1, k -CCDS, to decide if a node remains a non-backbone node, we need to test if every combination of its neighbors can reach one another through the backbone satisfying the three constraints. Let $n \in N(u)$ be the first neighbor and $m \in N(u)$ be any other neighbor of u . If k alternate disjoint paths do not exist between nodes m and n , then the backbone is not fully constructed yet and hence u becomes a backbone node. If k alternate disjoint paths exist between nodes m and n :

Case 1: Assume n is a backbone node and m is a non-evaluated non-backbone node or a backbone node. We need to check if k constrained paths exist between nodes m and n . Since k alternate disjoint paths exist and since k -CCDS can be constructed, then there exists k constrained paths between nodes m and n .

Case 2: Let n and m be non-evaluated nodes where both have lower priority than u . Since k alternate disjoint paths exist between nodes m and n satisfying the constraints, then we move to the next non-evaluated neighbor $p \in N(u)$. However, since we have a full mesh network and all edges are equally weighted, p is an exact replica of n and so are all the remaining non-evaluated nodes. Thus it is enough to test m and n only rather than all combinations with m . Once done with the first neighbor m , we need to choose n as our evaluation node and check it with p and all other remaining nodes. However, n is also an exact replica of m and thus we will obtain the same results. Therefore, it is enough to test if m and n have k constrained paths with one another because if it exists, it exists for all remaining non-evaluated node combinations.

Therefore, when k alternate constrained disjoint paths exist between nodes m and n , then we have a fully constructed backbone. Furthermore, since all non-backbone and backbone combinations satisfy the constraints and a single non-evaluated non-backbone constraint satisfaction is implied to all remaining combinations, we can mark node u as a non-backbone node without evaluating the remaining combinations. Thus, when the backbone is fully constructed it is enough to test one combination out of $(N - \text{size}(\text{backbone})) \times N^2$ combinations to know that all remaining nodes are non-backbone nodes. ■

Chapter 6

Experimental Results

The algorithms described in Chapter 4 have been implemented using MATLAB, executed using multi-threading and tested thoroughly for correctness and efficiency. Furthermore, a testing algorithm has been developed, Algorithm 6, to validate the correctness of the results that have been generated by the system algorithms.

6.1 Network Architecture

6.1.1 Scalable IPsec Networks

The logical connection between two hosts is identified by 2 Security Associations (SA) at each host, where one is used for inbound traffic and another for outbound traffic assuming that one protocol is being used i.e. ESP or AH. If ESP and AH are used together, then we need 2 SAs for each protocol and hence a total of 4 SAs at each node. Since IPsec tunnels are virtual links and every node can communicate securely through the overlay network, we can represent a network as a full mesh where each link represents a tunnel or an SA. Therefore we represented IPsec gateways as nodes and the SAs as undirected edges for every combination of nodes. In all our simulations, we assumed that only one protocol is used and the weight of an edge is taken to be equal to one, which means intermediary routers do not exist between two communicating parties. Additionally, since the degree of all the nodes are equal, we assigned unique IDs to the nodes and used it as our priority. However, the algorithm allows us to set the weight to any positive value.

6.1.2 General Scalable Networks

General scalable networks consist of computers or servers that are connected to each other through routers, switches, hubs etc. where the complexity of management and effective scalability increase with the increase in the number of users in

the network. To test our k -CCDS algorithm on general networks, we considered the nodes of our graph to represent routers, and the weight of the undirected edges as the number of intermediary hops needed to reach from one router to the other. For our simulations, we considered the weight of all edges to be equal to one. Since routers are complex devices that provide different functionalities, they have a limitation on the number of connections that they can handle efficiently. In realistic networks, most of the routers consist of 8 to 16 physical connections. The priority of nodes is based on the degree of each node in G , where the edges are uniformly distributed between 8 and 16.

6.2 Scalable IPsec Experimental Results

6.2.1 Simulation Parameters & Algorithm

The experiments are carefully designed to test the importance of the improvements introduced by our proposed k -CCDS algorithm while depicting realistic networks as much as possible. Furthermore, a solution exists for the tested scenarios and hence there was no need to relax the degree and/or path constraints. To test the effect of different parameters, we varied one of the parameters and fixed the remaining ones. The parameters that we have used along our experiments are summarized in the following table:

Parameter Name	Range of Values
Number of Nodes	10-1000
Number of Alternate Paths k	1-4
Path Bound m	2-7
Degree Constraint d	$\frac{N}{4}$ to $(N - 1)$
Number of Edges from a Node	N-1
Weight of an Edge	1

Table 6.1: Simulation Parameters for IPsec Networks

The path bound is taken to be a maximum of 7 in all experiments to prevent the effect of latency due to the presence of encryption and decryption between every two tunnel end-points.

The following algorithm, Algorithm 5, has been used to generate all the results that are included in this section. It calls k -CCDS to find the total number of SAs needed to construct a scalable IPsec network. Additionally, it calls Algorithm 6 to test the k -CCDS satisfaction of the subgraph generated from G :

Algorithm 5 Simulation Algorithm for Scalable IPsec Networks

Input: Parameters stated in Table 6.1.

Output: Total number of needed SAs, backbone size and k -CCDS subgraph.

- 1: Generate graph G which is a full mesh, based on the number of nodes, edge weight, and number of edges from a node.
- 2: Check if the generated graph G is connected by checking if its diameter is not ∞ . If it is not connected, then return an error.
- 3: Call k -CCDS(G, d, k, m)
- 4: $G_B = G$ // BackboneGraph G_B will contain the edges that will be included as part of the final k -CCDS subgraph
- 5: // Construct k -CCDS subgraph
- 6: **for** $\forall v \in G$ **do**
- 7: **for** $\forall u \in G$ where $u \neq v$ **do**
 // Remove connections between non-backbones nodes
- 8: **if** $e_{u,v} \in E$ **and** u and v are non-backbones **then**
- 9: Remove the edge $e_{u,v}$ from G_B
- 10: **end if**
 // Remove unused connections between backbone nodes
- 11: **if** $e_{u,v} \in E$ **and** u & v are backbones **and** $e_{u,v} \notin$ any DCSPT **then**
- 12: Remove the edge $e_{u,v}$ from G_B which was marked in Algorithm 1 because it hasn't been used as part of any shortest path.
- 13: **end if**
- 14: **end for**
- 15: **end for**
- 16: SA=0 // Calculate total SAs used
- 17: **for** $\forall v \in G_B$ **do**
- 18: **if** v is a non-backbone node **then**
- 19: // 2 SAs for the non-backbone and 2 for the backbone it is connected to
- 20: SA=SA+4
- 21: **else**
- 22: **for** $\forall u \in G_B$ where $u \neq v$ **do**
- 23: **if** $e_{u,v} \in E_B$ **and** u & v are backbone nodes **then**
- 24: SA=SA+2
- 25: **end if**
- 26: **end for**
- 27: **end if**
- 28: **end for**
- 29: Run test Algorithm 6

Algorithm 5 starts by generating a full mesh graph, and calls the k -CCDS based on the inputted parameters to return the nodes that have been chosen as a backbone in addition to the fixed pathbound and degree constraint if they have

been relaxed (lines 1-3). The backbone status of nodes is then used to remove the edges that exist between non-backbone nodes (lines 8-9) and the links that exist between backbone nodes given that they haven't been used by any path to provide constrained connectivity (lines 11-12). After obtaining our k -CCDS subgraph, we need to calculate the total number of SAs in our scalable IPsec network. For non-backbone nodes, since they use only one link out of k possibilities, we have 2 SAs for it and 2 SAs for the backbone node it is connected to (lines 18-20). As for the backbone nodes, if it is connected to another backbone node then we add 2 SAs for this former backbone node (lines 22-26). Hence, we obtain our k -CCDS architecture and the total number of SAs required by the scalable IPsec network. Finally, we can run the test algorithm to make sure that all nodes are connected and, degree and path constraints are satisfied when $0,1,\dots,k-1$ nodes fail simultaneously.

6.2.2 Test Algorithm

Algorithm 6 TestBackboneConnectivity($G_B, \mathbf{d}, m, \mathbf{B}$)

Input: Backbone graph $G_B(V, E_B)$, degree constraint \mathbf{d} , path bound m , and nodes that have backbone status \mathbf{B}

Output: If constructed k -CCDS passes the test, return **True**. Else return **False** with the error message.

```

1: for  $r=0,1,\dots,k-1$  do
2:   for every combination of  $r$  nodes do
3:     Temporarily remove the combination of  $r$  nodes when  $r > 0$ 
4:     for every remaining node  $v$  do
5:       Call DCSPT( $G, \mathbf{d}, v, \mathbf{B}$ )
6:       if DCSPT returns violating node  $p$  then // degree( $p$ ) >  $d_p$ 
7:         Return False with the following error message "Degree Constraint
           cannot be satisfied at violating node  $p$  when DCSPT is run from  $v$ "
8:       end if
           // Test if every node is reachable
9:       for every remaining node  $u$  where  $u \neq v$  do
10:        distance=DCSPTdistance( $v, u$ );

```

```

11:         if distance==∞ or distance > m then
12:             Return False with the following error message “Path Bound
                cannot be satisfied at node  $u$  starting from node  $v$ ”
13:         end if
14:     end for
15: end for
16: end for
17: return True
18: end for

```

6.2.3 Algorithm Efficiency

The running time of DCSPT, Algorithm 2, on full mesh networks increases exponentially as the number of nodes in the network increases. Furthermore, for the evaluation of non-backbone nodes once the backbone is constructed and to decide that a node remains a non-backbone node, DCSPT is run on $N - M$ non-backbone nodes $(N - 1)^2$ times because we need to test if every combination of neighbors of a node has k constrained paths, where M represents the number of backbone nodes. To improve the performance of our algorithm on large networks, we applied Theorem 3 from Chapter 5 on k -CCDS (Algorithm 1) and thus decreased the number of times DCSPT is called for the evaluation of non-backbone nodes from $(N - M) \times (N - 1)^2$ to 1.

6.2.4 Experimental Results

In our first scalable IPsec experiment, we varied the number of nodes between 10 and 1000 and the edges from each node was fixed to be equal to $N-1$ because of the full mesh nature of the overlay IPsec network. As for our input parameters, we fixed the degree constraint $d=N-1$ and the path bound $m=2$, and varied k between 1 and 4 (from no failure toleration to the toleration of at least 3 failures). This experiment provides us with the best results because the degree constraint was relaxed to the maximum, however the path bound was minimized which is a highly desirable feature.

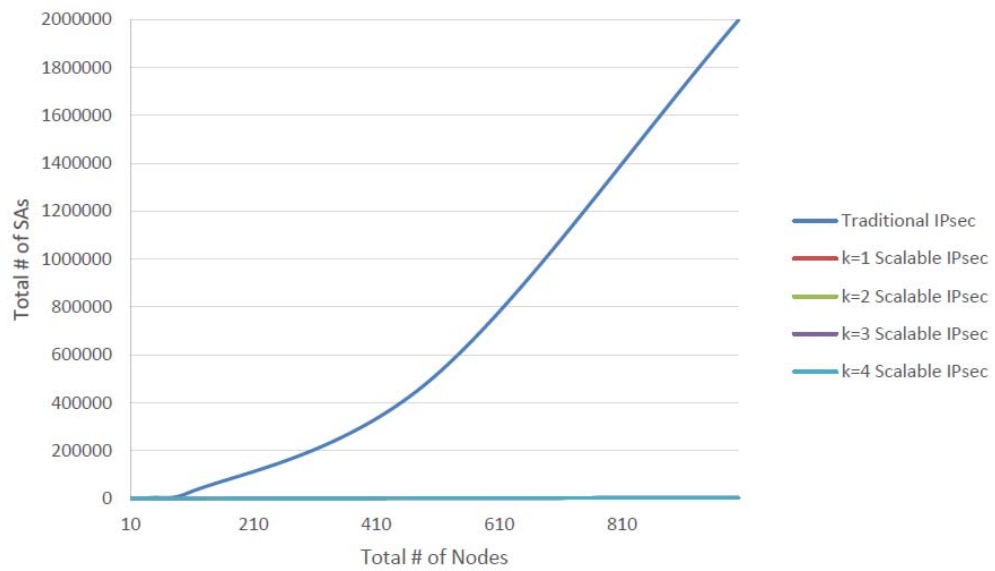


Figure 6.1: Experiment 1 - Scalable IPsec SA Improvement

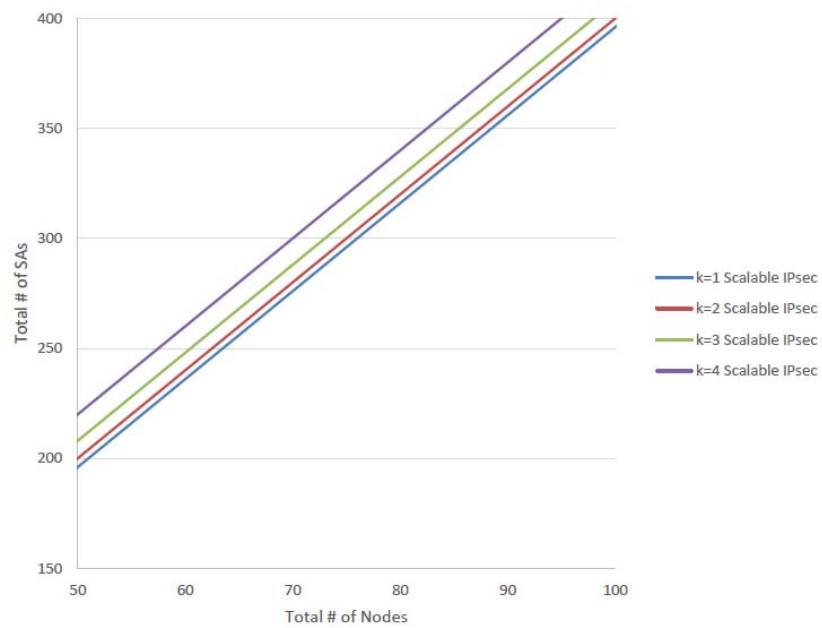


Figure 6.2: Experiment 1 - Scalable IPsec SA Improvement Scaled

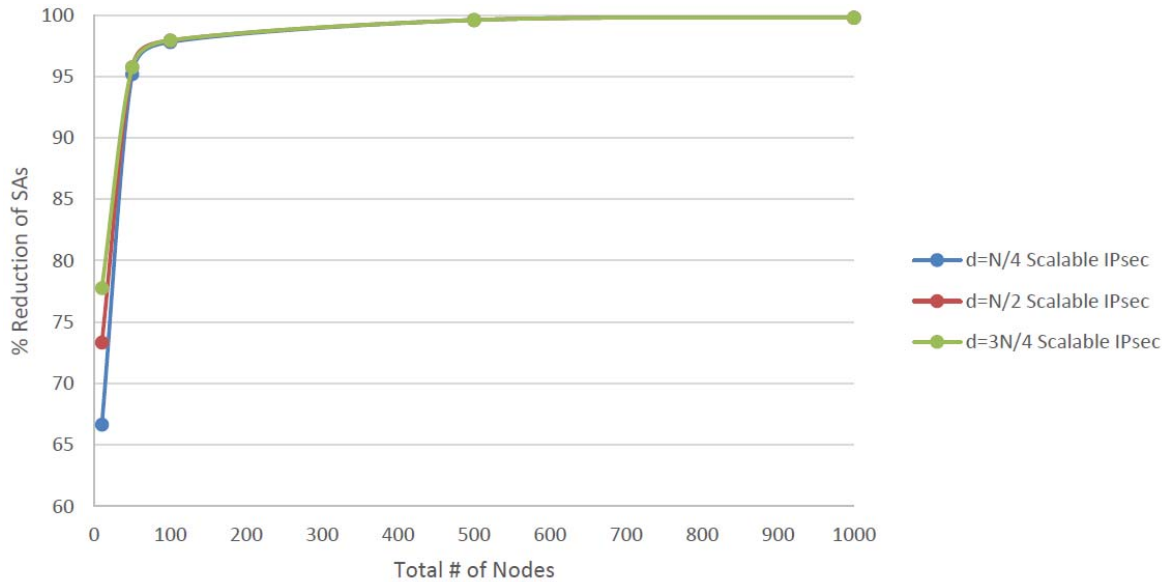


Figure 6.3: Experiment 1 - Scalable IPsec % Reduction of SAs

As we observe in Figures 6.1 and 6.2, the number of SAs have been reduced vastly from millions to a couple of thousands. For example, for a network consisting of 1000 nodes when $k=4$, the number of IPsec SAs have been reduced from 1,998,000 in traditional IPsec networks to 4020 in our scalable IPsec architecture. Furthermore, any node can reach to any other node in the network by using a maximum of 2 SAs. The detailed results are presented in Table 6.2:

$N \setminus k$	1	2	3	4
10	36/180	40/180	48/180	60/180
50	196/4900	200/4900	208/4900	220/4900
100	396/19800	400/19800	408 /19800	420/19800
500	1996/499000	2000/499000	2008/499000	2020/499000
1000	3996/1998000	4000/1998000	4008/1998000	4020/1998000

Table 6.2: Total Number of SAs of Experiment 1

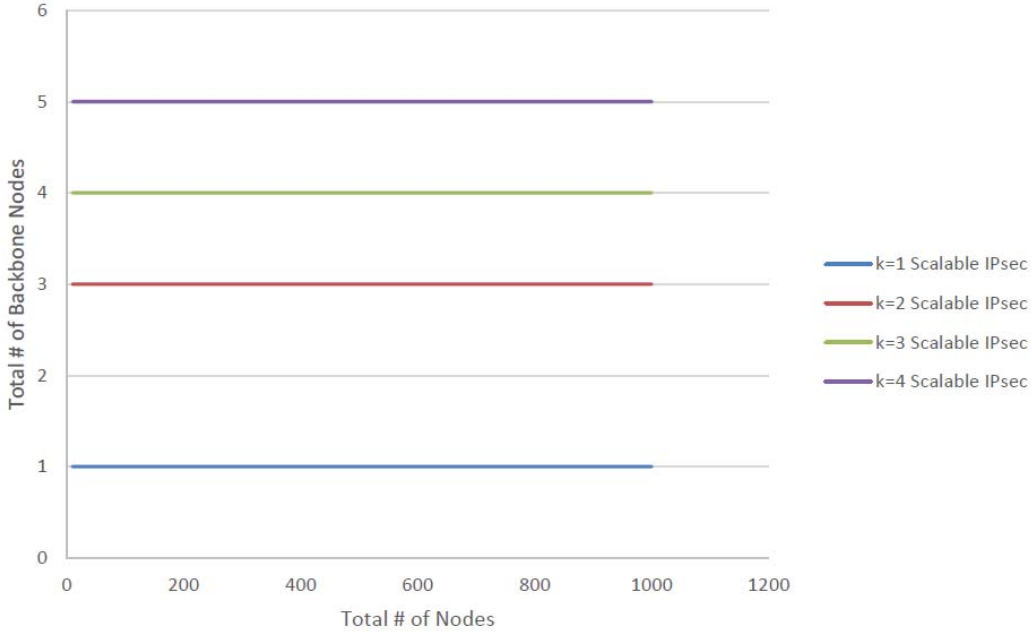


Figure 6.4: Experiment 1 - Scalable IPsec Number of Backbones

As for the number of backbone nodes shown in Figure 6.4, we have a minimal increase in the size of the backbone while adding redundancy in the network, which is a beneficial trade-off based on the results shown in the figure. For example, the size of the backbone has increased from 1 to 5 when k was increased from 1 to 4.

In our second experiment, we varied the number of nodes between 10 and 1000 and the edges from each node was fixed to be equal to $N-1$ because of the full mesh representation of IPsec networks. As for our input parameters, we fixed $k=2$ and the path bound $m \leq 7$, and varied the degree constraint d between $\frac{N}{4}$, $\frac{N}{2}$ and $\frac{3N}{4}$. For $d = \frac{N}{4}$ a solution exists when $m > 4$ and for the remaining cases a solution exists when $m > 2$. The reason why we need $m > 4$ is because every node can have a maximum of $\frac{N}{4}$ SAs, and since the network has N nodes to provide full IPsec connectivity, 4 backbone nodes are needed to provide connectivity to the whole network and thus the need of 3 SAs in between. Furthermore, when two non-backbone nodes need to communicate, since each has a connection to a backbone node, the total cost of the path will be equal to 5, thus the reason for $m > 4$.

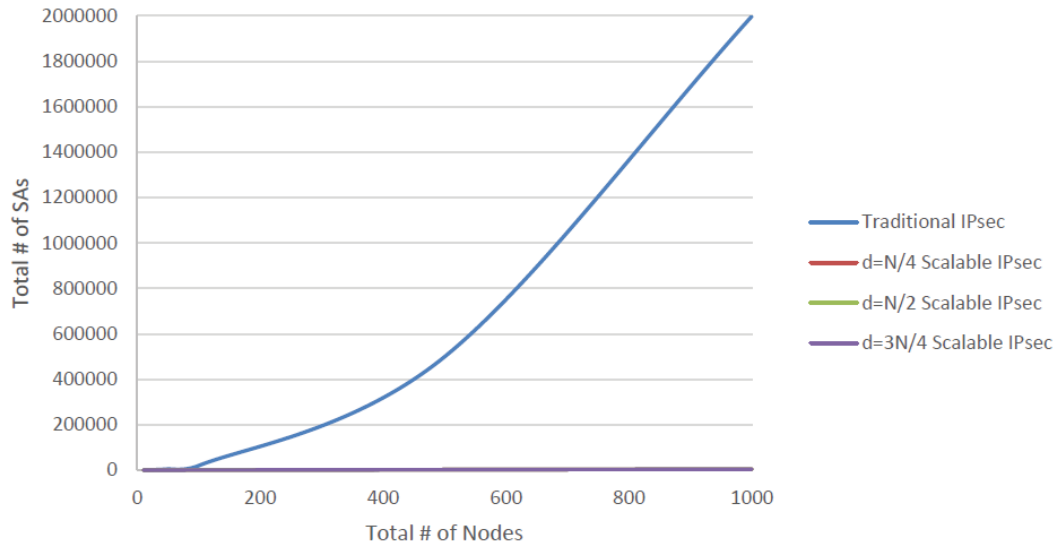


Figure 6.5: Experiment 2 - Scalable IPsec SA Improvement

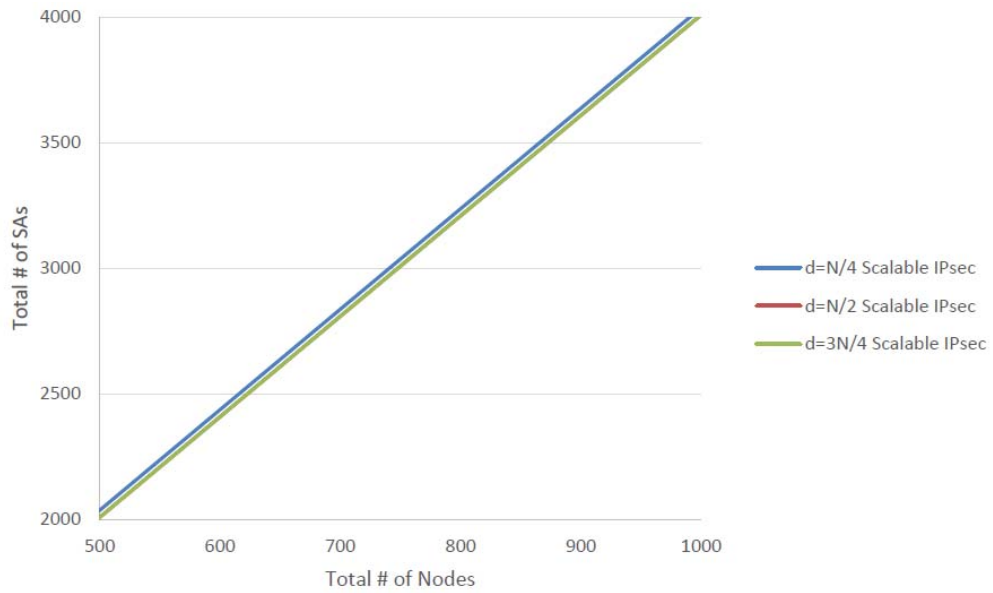


Figure 6.6: Experiment 2 - Scalable IPsec SA Improvement Scaled

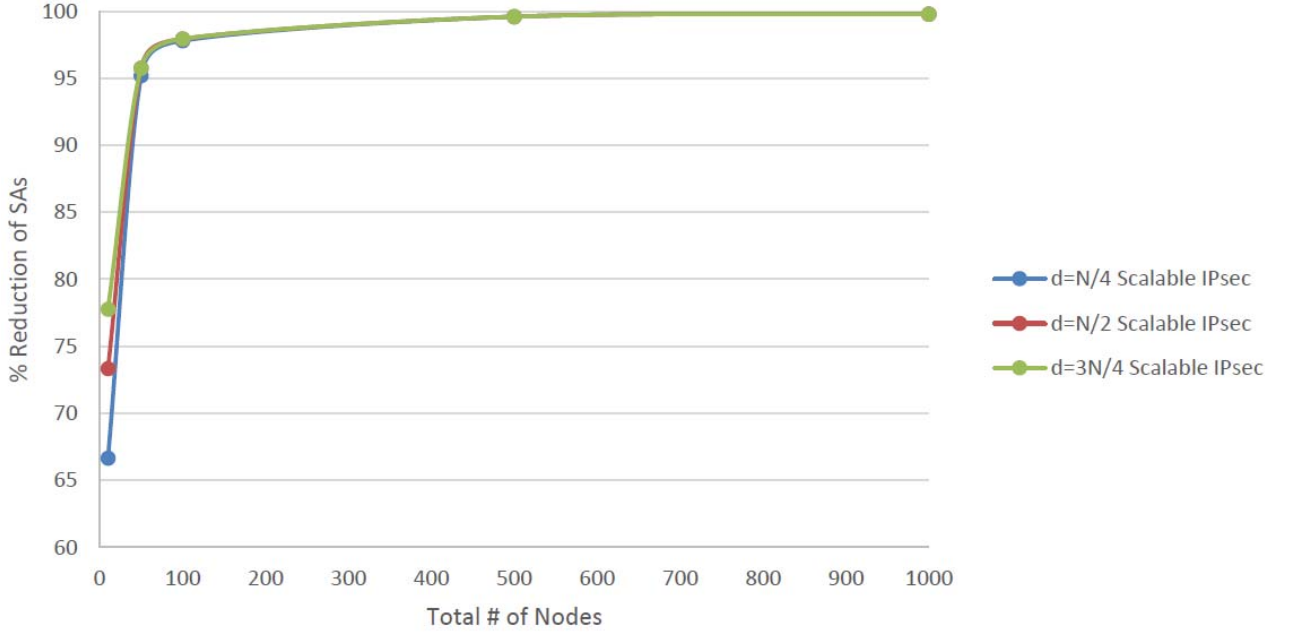


Figure 6.7: Experiment 2 - Scalable IPsec % Reduction of SAs

As observed in Figures 6.5 and 6.6, we obtained results very close to Experiment 1 which represents the best case scenario. To make Experiment 2 more realistic from the networking point of view, we varied the degree constraint d . Although m was increased for different degree constraints to find a solution, in both cases it was bounded above by 7 and hence the latency remains minimal. For the case $d = \frac{N}{4}$ we can use $m = 5$ and for the remaining two cases we can use $m = 3$. Furthermore, the total number of SAs remains minimal approaching the results of the first experiment. We can conclude that scenarios with realistic input parameters provide results that are very close to the ideal case scenario. The detailed results are represented in Table 6.3:

$N \setminus d$	$\frac{N}{4}$	$\frac{N}{2}$	$\frac{3N}{4}$
10	60/180	48/180	40/180
50	236/4900	208/4900	208/4900
100	436/19800	408/19800	408 /19800
500	2036/499000	2008/499000	2008/499000
1000	4036/1998000	4008/1998000	4008/1998000

Table 6.3: Total Number of SAs of Experiment 2

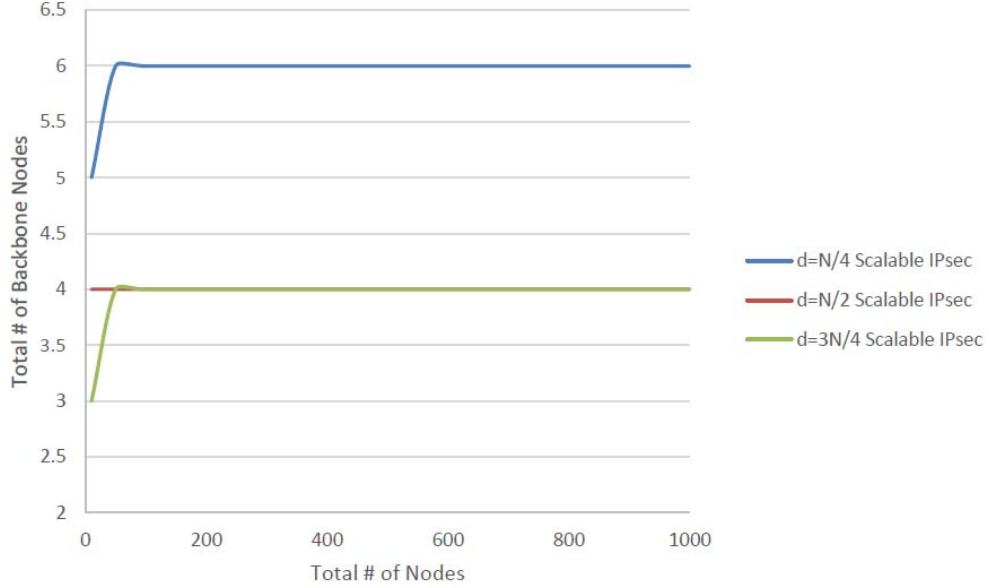


Figure 6.8: Experiment 2 - Scalable IPsec Number of Backbones

As for the number of backbone nodes shown in Figure 6.8, the maximum which is six is reached when $\mathbf{d}=\frac{N}{4}$ which is still minimal compared to 1000 nodes.

Analyzing the results of both experiments show that we were able to reduce the total number of SAs by 67% when $N = 10$ $\mathbf{d}=\frac{N}{4}$ and up to 99.8% when $N = 1000$ and $\mathbf{d}=\frac{3N}{4}$ as observed in Figures 6.3 & 6.7. It can also be deduced that the increase in the size of the network has no effect on the number of backbone nodes needed to have a k -CCDS of G , and the nodes with higher priorities formed the backbone of our network. The path bound m is the minimal possible in traditional IPsec networks by being equal to one which is the main reason why we have a very large number of SAs. The proposed algorithm increases the path bound m minimally to have a control over the latency and in return provides a smaller number of total IPsec SAs by forming a backbone of IPsec gateways. The degree constraint \mathbf{d} in traditional IPsec networks is always $N - 1$, whereas in our design it is variable and we have seen through Experiment 2 that setting it as $\frac{N}{4}$ will allow m to be less than 7 and also obtain a small backbone. As for the alternate disjoint paths that add redundancy, we have at least one in traditional IPsec networks because it depends in the way the underlying physical network is connected, whereas k -CCDS guarantees that the overlay network has at least $k - 1$ node disjoint paths. We can summarize and compare k -CCDS to current IPsec networks with the following table:

	Traditional IPsec	Scalable IPsec
Alternate Paths k	At least 1	1,2,3,4,..., k
Degree Constraint d	Fixed $N - 1$	Variable 2,5,10,..., $N - 1$
Path Bound m	1	minimize by setting $2 \leq m \leq 7$

Table 6.4: Traditional IPsec vs. Scalable IPsec

To calculate the expected total number of SAs, we first recall the following theorem from [2]:

Theorem: The expected number of backbone nodes selected by the k -coverage condition is $O(k)$ times the size of a minimal k -CDS.

Let n represent the total number of nodes in a network and NB the expected number of non-backbone nodes such that

$$NB = n - \text{sizeof}(\text{min } k\text{-CDS}) \times O(k)$$

Hence, the expected number of SAs that have been removed between backbone and non-backbone nodes is $4 \sum_{i=1}^{NB} (\text{degree}(v_{\text{non-backbone},i}) - 1)$. The expected number of SAs that are removed between backbone nodes is a function of the number of backbone nodes because it depends on the links that haven't been marked in Algorithm 1. Thus the # of unmarked links is a function of backbone nodes eg. $f(\text{backbone})$. Therefore, the expected total number of SAs in a k -CCDS subgraph is:

$$2 \sum_{i=1}^n \text{degree}(v_i) - 4 \sum_{i=1}^{NB} (\text{degree}(v_{\text{non-backbone},i}) - 1) - 4 \times \text{\#ofunmarkedlinks}$$

6.3 General Scalable Networks

6.3.1 Simulation Parameters & Algorithm

The number of nodes and edges used in our experiments represent realistic general networks as much as possible. In general network, the number of physical links from a router varies between 8 and 16, and thus for our experimental scenarios we varied it uniformly between 8 and 16. As for the number of nodes we varied it between 50 and 500 to show the improvements that our algorithm introduces in large scalable networks. Furthermore, a solution exists for the tested scenarios and hence there was no need to relax the degree and/or path constraints. The parameters that we have used along our experiments are summarized in the following table:

Parameter Name	Range of Values
Number of Nodes	50-500
Number of Alternate Paths k	1,2
Path Bound m	10-20
Degree Constraint d	10,12,14,16
Number of Edges from a Node	8-16
Weight of an Edge	1

Table 6.5: Simulation Parameters for General Networks

Algorithm 7 uses the algorithms described in Chapter 4, and hence is very similar in construct to Algorithm 5. What differs is the generation of the graph which is random, and the procedure of calculating the number of links needed to construct scalable general networks. Furthermore, it calls Algorithm 6 to test the k -CCDS satisfaction of the subgraph generated from G :

Algorithm 7 Simulation Algorithm for Scalable General Networks

Input: Parameters stated in Table 6.5.

Output: Total number of needed links, backbone size and k -CCDS subgraph.

- 1: Generate a random graph G based on the number of nodes, edge weight, and number of edges from a node.
 - 2: Check if the generated graph G is connected by checking if its diameter is not ∞ . If it is not connected, then return an error.
 - 3: Call k -CCDS(G, d, k, m)
 - 4: $G_B = G$ // BackboneGraph G_B will contain the edges that will be included as part of the final k -CCDS subgraph
 - 5: // Construct k -CCDS subgraph
 - 6: **for** $\forall v \in G$ **do**
 - 7: **for** $\forall u \in G$ where $u \neq v$ **do**
 - 8: // Remove connections between non-backbones nodes
 - 9: **if** $e_{u,v} \in E$ **and** u and v are non-backbones **then**
 - 10: Remove the edge $e_{u,v}$ from G_B
 - 11: **end if**
 - 12: // Remove unused connections between backbone nodes
 - 13: **if** $e_{u,v} \in E$ **and** u & v are backbones **and** $e_{u,v} \notin$ any DCSPT **then**
 - 14: Remove the edge $e_{u,v}$ from G_B which was marked in Algorithm 1 because it hasn't been used as part of any shortest path.
 - 15: **end if**
 - 16: **end for**
 - 17: **end for**
-

```

16: Links=0 // Calculate total links used
17: for  $\forall v \in G_B$  do
18:   if  $v$  is a non-backbone node then
19:     // A link between the non-backbone and a backbone node.
20:     Links=Links+1
21:   else
22:     for  $\forall u \in G_B$  where  $u \neq v$  do
23:       if  $e_{u,v} \in E_B$  and  $u$  &  $v$  are backbone nodes then
24:         Links=Links+1
25:       end if
26:     end for
27:   end if
28: end for
29: Run test Algorithm 6

```

6.3.2 Experimental Results

In the first scalable general networks experiment, the number of nodes was varied between 50 and 500, and the edges uniformly between 8 and 16. The aim of this experiment is to check the improvements of the algorithm that we obtain through varying the degree constraint. Hence we fixed the path bound $m=20$, the alternate disjoint paths $k=2$, and we varied the degree constraint d between 10 and 16. Since we are randomly creating our graph based on our input parameters, we repeated the simulations 10 times for every combination of input parameters.

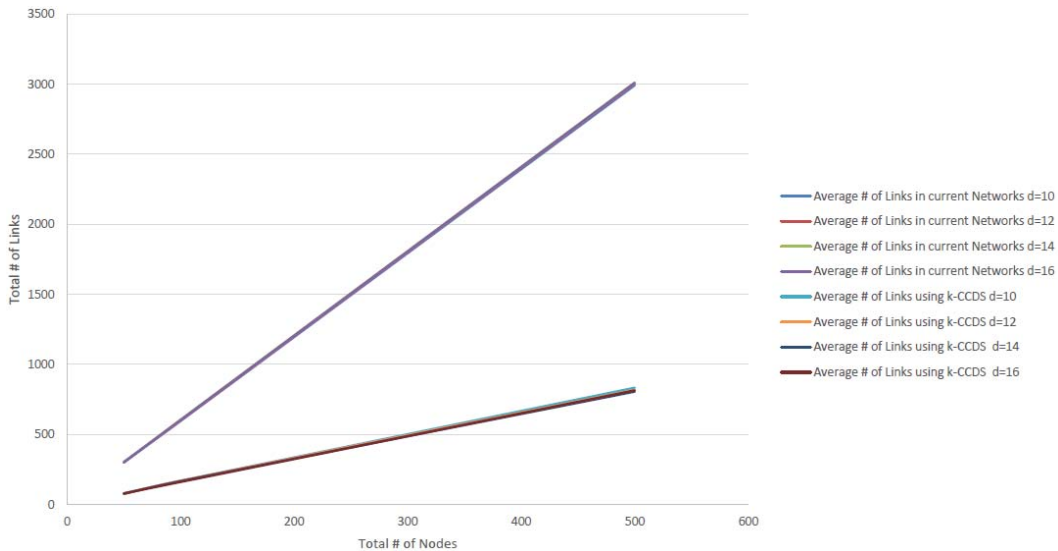


Figure 6.9: Experiment 1 - General Networks Links' Improvement

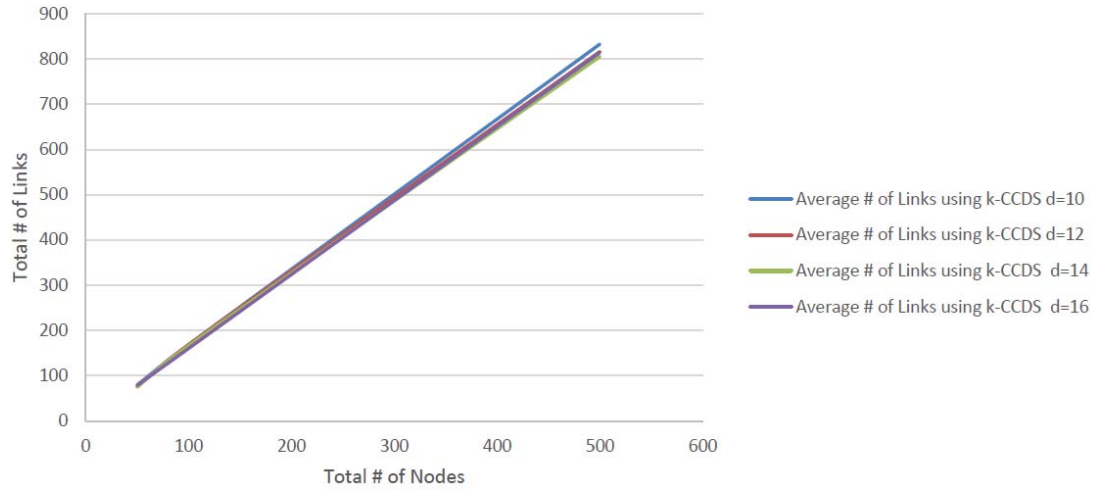


Figure 6.10: Experiment 1 - General Networks Links' Improvement Scaled

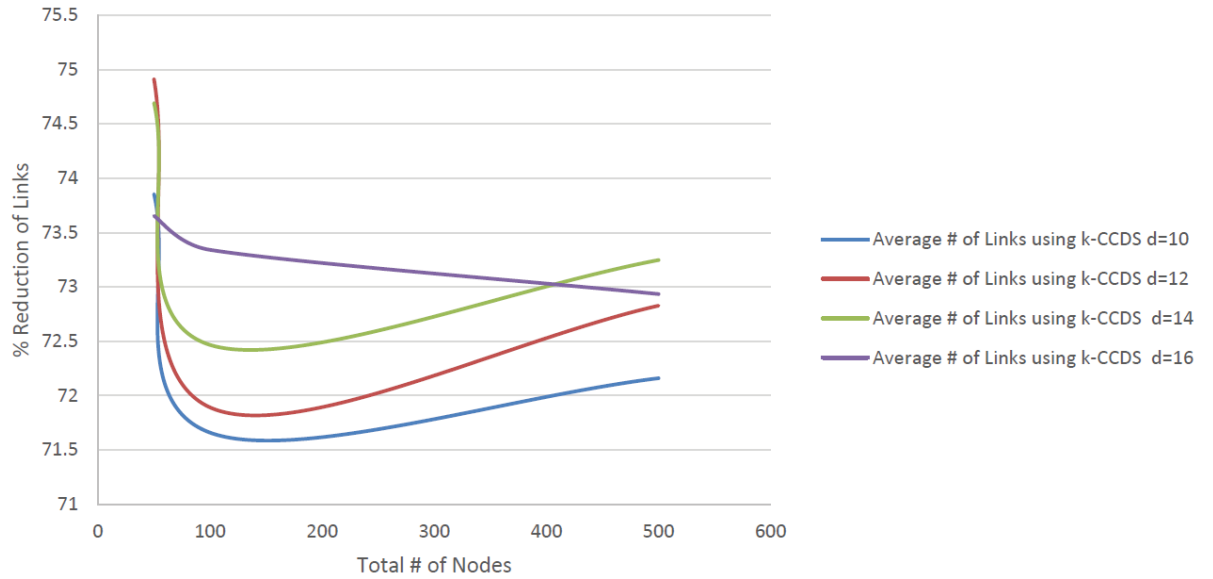


Figure 6.11: Experiment 1 - General Networks % Reduction of Links

As we can observe in Figures 6.9 and 6.10, the number of links required has been reduced from thousands to a couple of hundreds. For example, for a network consisting of 500 nodes where $d=12$, the number of links has been reduced from 2991 to 791. The detailed results are presented in Table 6.6:

$N \setminus d$	10	12	14	16
50	79.3/303.3	75.3/300.1	76/300.3	79.2/300.6
100	169/596.3	168.6/599.8	166.5/604.7	160.5/602.03
500	833/2992	816.5/3004.8	805.5/3010.8	814.4/3008.9

Table 6.6: Total Number of Links of Experiment 1

$N \setminus d$	10	12	14	16
50	4.3982	6.0194	6.4979	6.4601
100	9.798	10.6583	16.2429	12.4833
500	36.046	33.6961	23.2959	34.7121

Table 6.7: Standard Deviation of Scalable Network Links of Experiment 1

$N \setminus d$	10	12	14	16
50	10.2529	10.0824	10.7295	8.0581
100	9.0068	13.0196	8.3673	12.85
500	20.2649	15.6274	27.8239	29.9275

Table 6.8: Standard Deviation of Traditional Network Links of Experiment 1

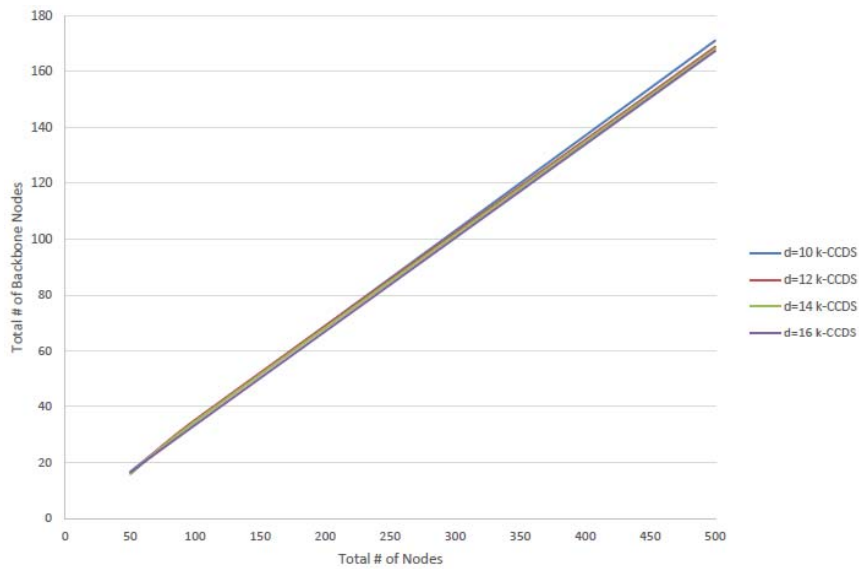


Figure 6.12: Experiment 1 - General Networks Number of Backbones

$N \setminus d$	10	12	14	16
50	1.0593	1.2867	1.2867	0.8489
100	1.3984	2.2136	2.3594	1.7159
500	4.2201	7.0597	4.2151	5.7966

Table 6.9: Standard Deviation of Number of Backbones of Experiment 1

As for the number of non-backbone nodes, we can observe in Figure 6.12 that the number of backbone nodes represents only 35% of the total number of nodes which is small taking into consideration the small number of physical links available at each node (8 to 16).

In our second experiment, we varied the number of nodes between 50 and 500, and the number of edges uniformly between 8 and 16 as in Experiment 1. As for the other input parameters, we fixed the path bound $m=20$ and the degree constraint $d=0.5E=12$, and varied k between 1 and 2. In general scalable networks having a single alternate path is considered to be enough. As in Experiment 1, we repeated every simulation 10 times due to random generation of graphs.

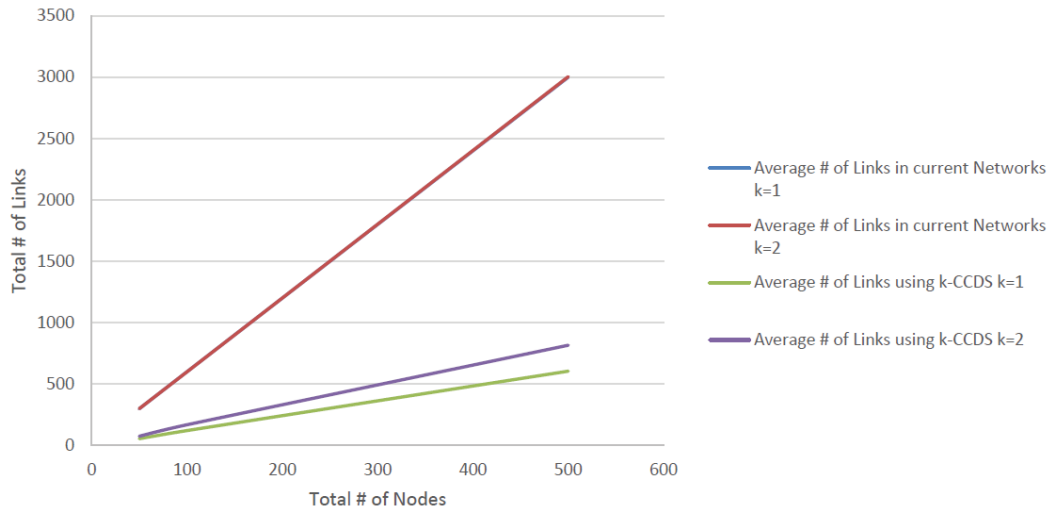


Figure 6.13: Experiment 2 - General Networks Links' Improvement

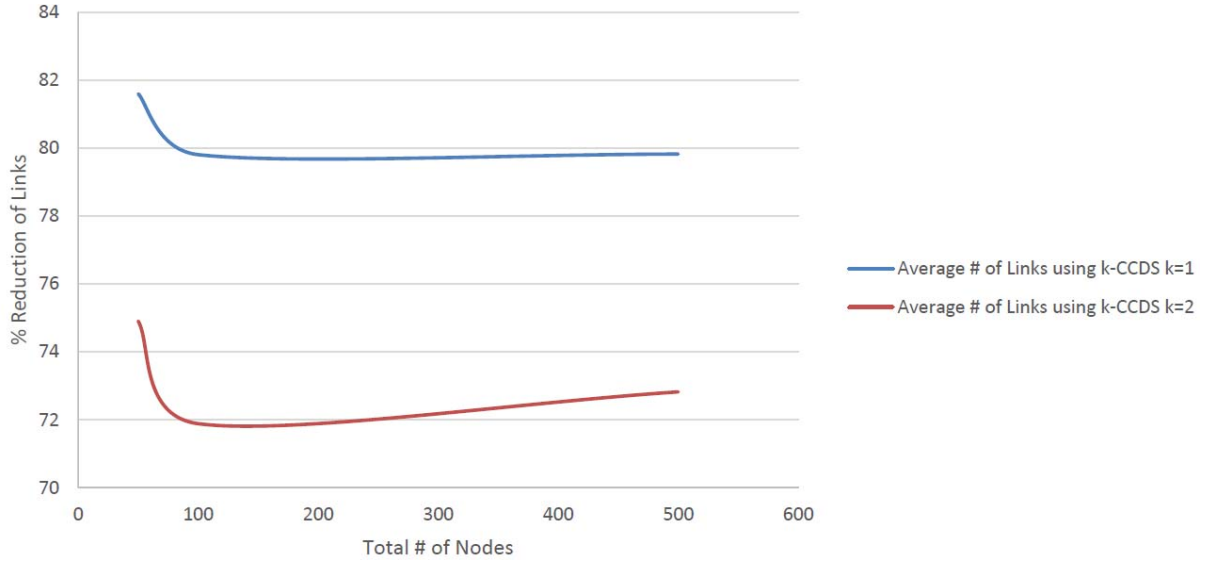


Figure 6.14: Experiment 2 - General Networks % Reduction of Links

We can see through Figure 6.13 that we have reduced the number of links needed by at least 70%. It is logical to have the number of links needed for $k = 2$ to be larger than $k = 1$ because in the former case we need to have an alternate path between any two nodes, while in the latter case we do not have any redundancy in the network. Although we do not have any redundancy constraint for $k = 1$, we can still observe the improvement introduced by $k - CCDS$ which reach up to 81%. The results observed can be summarized in Table 6.10:

$N \setminus k$	1	2
50	55/298.9	75.3/300.1
100	121.2/600.2	168.6/599.8
500	605.2857/3000.1	816.5/3004.8

Table 6.10: Total Number of Links of Experiment 2

$N \setminus k$	1	2
50	4	6.0194
100	7.0993	10.6583
500	18.7146	33.6961

Table 6.11: Standard Deviation of Scalable Network Links of Experiment 2

$N \setminus k$	1	2
50	5.9526	10.0824
100	12.3989	13.0196
500	34.0804	15.6274

Table 6.12: Standard Deviation of Traditional Network Links of Experiment 2

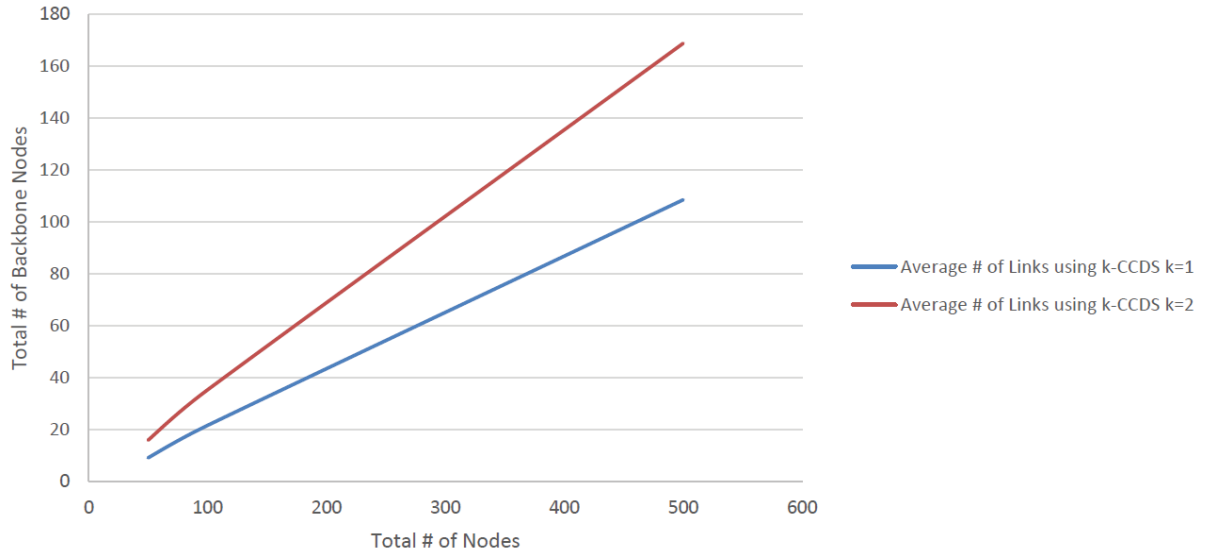


Figure 6.15: Experiment 2 - General Networks Number of Backbones

$N \setminus k$	1	2
50	0.9944	1.2867
100	2.1705	2.2136
500	5.127	7.0597

Table 6.13: Standard Deviation of Number of Backbones of Experiment 2

As for the number of backbone nodes shown in Figure 6.15, it constitutes a maximum of 32% of the total number of network nodes which is considered to be small.

Analyzing the results of both experiments show that we were able to reduce the total number of links by 71% when $N = 100$ $d=12$ and up to 80% when $N = 500$ and $k = 1$ as observed in Figures 6.11 and 6.14. It can also be observed

that the increase in the size of the network does not have a large effect on the number of backbone nodes needed to have a k -CCDS of G , and the nodes with higher priorities formed the backbone of our network. Furthermore, k -CCDS allows us to obtain k alternate disjoint paths in our general networks however it is enough to have one or two alternate paths, whereas in traditional general networks we cannot guarantee that we have alternate disjoint paths. As for the path bound it is similar to the path bound in realistic current general networks and hence we allowed it to be a maximum of 20 for large networks. We can summarize and compare k -CCDS to current general networks with the Table 6.14:

	General Networks	Scalable General Networks
Alternate Paths k	At least 1	$1,2,3,\dots,k$
Degree Constraint d	No constraint, Fixed 8,16	Variable 8, \dots ,16
Path Bound m	10-20	10-20

Table 6.14: General Networks vs. Scalable General Networks

Chapter 7

Conclusion and Future Work

In our research we designed and implemented k -CCDS that reduces the number of SAs needed compared to traditional IPsec networks by 66% to more than 99.5%. This results in the reduction of latency which is generated from the heavy cryptographic functions due to the presence of large number of SAs, reduction of memory required to store the SAs and SPs, and finally minimizing configuration overheads for key exchanges. The proposed architecture is not only applicable to IPsec networks, but also to any type of networks that requires the satisfaction of certain constraints i.e. bounds on number of devices on nodes, bound on the cost of a path and improving network redundancy. We can thus conclude that, k -CCDS is a general algorithm that constructs a k connected network consisting of backbone and non-backbone nodes which can sustain a failure of at least $k - 1$ nodes. When a node wants to communicate with any other node, the paths that are used for transmission are the shortest costing paths and the number of connections (degree) are equally balance between all backbone nodes. The results that we have obtained in Chapter 6 confirm the improvements that k -CCDS introduces in the construction of both scalable IPsec networks and general scalable networks. Furthermore, if a k -CCDS cannot be constructed with the given input parameters, the algorithm will find the relaxed degree and path constraints, which can be further used as the new input parameters to construct a k -CCDS.

Future work includes improving the efficiency of DCSPT algorithm , re-establishing new alternate paths when failures occur, dynamically adding new nodes that want to join the network, and finally constructing a k -CCDS network consisting of smaller k -CCDS networks.

Appendix A

Abbreviations

AH	Authentication Header
DCSPT	Degree Constrained Shortest Path Tree
ESP	Encapsulating Security Payload
IKE	Internet Key Exchange
IPsec	IP security
k -CCDS	k Constrained Connected Dominating Set
k -CDS	k Connected Dominating Set
SA	Security Association
SADB	Security Association Database
SP	Security Policy
SPD	Security Policy Database
SPI	Security Parameter Index
SPT	Shortest Path Tree

Bibliography

- [1] J. Wu and H. Li, “On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks,” in *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, DIALM '99, (New York, NY, USA), pp. 7–14, ACM, 1999.
- [2] F. Dai and J. Wu, “On Constructing k -Connected k -Dominating Set in Wireless Networks,” in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 81a–81a, April 2005.
- [3] S. Kent and R. Atkinson, “Security Architecture for the Internet Protocol.” RFC 2401, November 1998.
- [4] Doraswamy, Naganand and Harkins, Dan, *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall, 1999.
- [5] D. Harkins, D. Carrel, *et al.*, “The Internet Key Exchange (IKE),” tech. rep., RFC 2409, November 1998.
- [6] D. Maughan, M. Schertler, M. Schneider, and J. Turner, “Internet Security Association and Key Management Protocol.” RFC 2408, November 1998.
- [7] H. Orman, “The OAKLEY key determination protocol.” RFC 2412, November 1998.
- [8] H. Krawczyk, “SKEME: a versatile secure key exchange mechanism for Internet,” in *Network and Distributed System Security, 1996., Proceedings of the Symposium on*, pp. 114–127, February 1996.
- [9] O. Adeyinka, “Analysis of problems associated with IPSec VPN Technology,” in *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*, pp. 001903–001908, May 2008.
- [10] F. Palmieri, “VPN scalability over high performance backbones evaluating MPLS VPN against traditional approaches,” in *Computers and Communication, 2003. (ISCC 2003). Proceedings. Eighth IEEE International Symposium on*, pp. 975–981 vol.2, June 2003.

- [11] P. Ferguson and G. Huston, “What is a VPN?.” Cisco Systems, Tech. Rep., 1998.
- [12] E. C. Rosen and Y. Rekhter, “Bgp/MPLS VPNs.” RFC 2547, March 1999.
- [13] J. D. Clercq and O. Paridaens, “Scalability implications of virtual private networks,” *IEEE Communications Magazine*, vol. 40, pp. 151–157, May 2002.
- [14] H. Ould-Brahim, B. Gleeson, G. Wright, T. Sloane, R. Bach, R. Bubenik, and A. Young, “Network based IP VPN architecture using virtual routers.” IETF draft, <https://tools.ietf.org/html/draft-ietf-13vpn-vpn-vr>, 2006, work in progress.
- [15] E. C. Rosen, J. De Clercq, O. Paridaens, Y. T’Joens, and C. Sargor, “Use of PE-PE IPsec in RFC2547 VPNs.” IETF draft, <https://draft-ietf-ppvpn-ipsec-2547--01.txt>, workinprogress, 2005, work in progress.
- [16] Juniper, “Group VPN Technology Overview.” http://www.juniper.net/techpubs/en_US/junos14.2/topics/concept/group-vpn-mx-overview.html, February 2015.
- [17] Juniper, “Group VPNv2 Technology Overview.” http://www.juniper.net/documentation/en_US/junos15.1/topics/concept/group-vpn-mx-overview.html, February 2016.
- [18] CISCO, “Group Encrypted Transport VPN (Get VPN) Design and Implementation Guide.” http://www.cisco.com/c/dam/en/us/products/collateral/security/group-encrypted-transport-vpn/GETVPN_DIG_version_1_0_External.pdf, March 2015.
- [19] S. C. Narula and C. A. Ho, “Degree-constrained Minimum Spanning Tree,” *Computers & Operations Research*, vol. 7, no. 4, p. 239249, 1980.
- [20] J. Clausen, “Branch and bound algorithms-principles and examples,” *Department of Computer Science, University of Copenhagen*, pp. 1–30, 1999.
- [21] J. Harant, A. Pruchnewski, and M. Voigt, “On Dominating Sets and Independent Sets of Graphs,” *Combinator. Probab. Comp. Combinatorics, Probability and Computing*, vol. 8, no. 6, p. 547553, 1999.
- [22] E. Boros, P. Heggernes, P. van’t Hof, and M. Milanič, “Vector connectivity in graphs,” *Networks*, vol. 63, no. 4, pp. 277–285, 2014.

- [23] F. Dai and J. Wu, “An extended localized algorithm for connected dominating set formation in ad hoc wireless networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, pp. 908–920, Oct 2004.
- [24] S. Saginbekov and I. Korpeoglu, “An energy efficient scatternet formation algorithm for Bluetooth-based sensor networks,” in *Wireless Sensor Networks, 2005. Proceedings of the Second European Workshop on*, pp. 207–216, Jan 2005.
- [25] S. Saginbekov, *Sensor and Ad Hoc Networks: Theoretical and Algorithmic Aspects*, ch. Forming Energy-Efficient Bluetooth Scatternets for Sensor Networks, pp. 297–306. Boston, MA: Springer US, 2008.
- [26] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*, vol. 6. MIT Press Cambridge, 2001.
- [27] K. Menger, “Zur allgemeinen kurventheorie,” *Fundamenta Mathematicae*, vol. 10, no. 1, pp. 96–115, 1927.
- [28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, March 2008.
- [29] Open vSwitch. openvswitch.org.
- [30] B. Pfaff and B. Davie, “The Open vSwitch Database Management Protocol.” RFC 7047, December 2013.
- [31] “Open vSwitch Database Schema.” <http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf>.